Developer Reputation Estimator: Increasing the Transparency of Developer Contributions in Open Source Software

A Dissertation Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Andrey Karnauch

May 2020

© by Andrey Karnauch, 2020

All Rights Reserved.

To Mama and Tato

Acknowledgments

I would like to first thank my advisor, Dr. Audris Mockus, for always making the impossible seem possible and challenging me to step out of my comfort zone. I would also like to thank all the members on our research team for the support they provided on this journey and for always providing thoughtful feedback on the work I was doing. I would also like the thank my committee members, Dr. Ruoti and Dr. Henley. To my family and friends, thank you for your unbounded amount of love and emotional support, something I learned to be equally as (if not more) important as the academic guidance I received. Finally, I would like to thank the EECS department at the University of Tennessee for enabling this opportunity.

As a note, this piece of work is a culmination of research I have done towards my Masters degree, and is adapted from some of my previous work, including [14] and [8].

Abstract

In this work, we present Developer Reputation Estimator (DRE), a web application that measures the technical and social aspects of a developer's expertise in the open source software (OSS) ecosystem. We provide several measures, from basic activity trace counts (e.g. number of commits, projects, etc.) to more advanced impact measures, such as tracking code re-use for a particular developer. We perform these measures on a novel scale using the World of Code infrastructure, which houses over 18 billion Git objects from many of the major OSS platforms. Together, these measures increase the transparency of a developer's experience in OSS and can be utilized to quickly establish trust between developers by gauging each other's skill sets and social network. As a byproduct of developers signing up for DRE, we also build a ground-truth dataset consisting of author identities we know belong to a single developer as well as the ones they do not claim. This dataset can be used as a training set for researchers to perform identity correction on author identities to better model the relationship between developers and other entities in OSS.

DRE is the first system that utilizes the WoC infrastructure in this way. It provides several use cases, including lowering the barrier of entry for developers attempting to contribute to a new OSS project, aiding researchers with a training dataset for identity correction, and allowing recruiters to corroborate resume claims made by applicants or seek out qualified developers based on a certain set of criteria. While this first version of DRE might not provide full support for each use case, we motivate these use cases and several other areas of future work to realize the full potential of our application.

Table of Contents

1 Introduction **1**

2 Pre-Processing Challenges **4**

2.1 Data collection 4

2.2 Data quality 5

2.2.1 Author Identities 6

2.2.2 Duplicate Projects 7

2.3 Data size 8

3 DRE: Walk-through and Design **9**

3.1 Usage 9

3.1.1 Sign-up and Search 9

3.1.2 Backend Processes 11

3.1.3 Upload list of IDs 13

3.2 Measures 13

3.2.1 Aggregated Contributions 15

3.2.2 Projects 16

3.2.3 Programming Languages 18

3.2.4 Collaborators 18

3.2.5 Torvalds Index 18

3.2.6 Blob Duplication 21

3.3 Software Architecture 23

4 Use Cases and Limitations **26**

4.1 Intended Users 26

4.1.1 Individual Developers 26

4.1.2 Researchers 27

4.1.3 Recruiters 27

4.2 Initial Feedback 28

4.3 Limitations 28

5 Related Work **30**

6 Conclusions **32**

6.1 Future Work 33

Bibliography **34**

Vita **39**

List of Figures

1.1 GitHub Activity Profile 2

2.1 The World of Code (WoC) Infrastructure Overview [15] 5

2.2 Two Identities (Andrey and zol0) for the Same Developer 7

3.1 The form a user is asked to fill out in order to locate potential author IDs for the user. Sample responses are shown. 10

3.2 The results page of potential author IDs that belong to a developer. The IDs a user recognizes are chosen (blue) and unfamiliar ones are left alone (white). 12

3.3 The front and back-end flow of DRE (from left to right). 14

3.4 The basic aggregation measures displayed at the top of a developer's profile. 15

3.5 A list of project's the developer has worked on along with their commit count vs. the total commit count. 17

3.6 A pie chart that breaks down the different programming language usage of a developer. 19

3.7 A breakdown of the projects a developer and collaborator (Audris Mockus in this case) have worked on. 20

3.8 Torvalds Index: The path between a developer and Linus Torvalds, with author IDs as nodes and projects as edges. 22

3.9 Blob Duplication Counts 23

3.10 The MERN stack as used by DRE 25

Chapter 1

Introduction

Modern software development provides an avenue for developers all over the world to collaborate on projects through the open source software (OSS) ecosystem. As this ecosystem continues to grow, developers may find themselves overwhelmed when seeking out collaborative opportunities with other developers. This challenge presents itself in many forms, including new developers looking to join one of the millions of projects in this ecosystem or a project maintainer seeking out one developer out of millions that best suits their needs. As a result, there exists a need for developers to be able to quickly establish trust between each other by shedding some light on their skill sets.

Previous work [16, 23] has shown that both technical and social factors can play an important role in building the trust between contributors and maintainers. Social factors, such as repeated interactions [12] between maintainers and contributors, are often the best way to establish trust and increase the chances of pull request acceptance [7, 9, 27] or issue resolution [11], both of which are ways developers can contribute on social coding platforms such as GitHub. However, the rapidly increasing pool of millions of developers and projects tests the scalability of the existing approaches to establish trust, due to the time cost associated with maintainers needing repeated interactions with every potential contributor. As a result, other, more efficient means of establishing trust are needed, such as trustworthy measures of a potential contributor's technical experience.

To answer this need, OSS platforms have implemented ways to quickly summarize the technical work developers have done in the past. GitHub, for example, provides developer

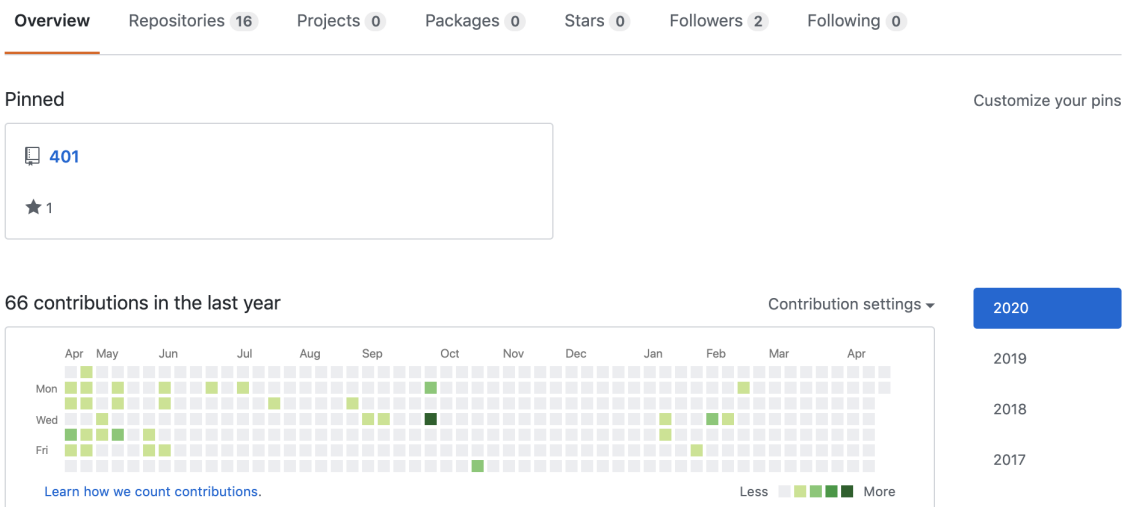

Figure 1.1: GitHub Activity Profile

profiles that show their activity over all projects on GitHub as seen in 1.1. Expertise Browser [19] was used to support globally distributed development by both showing the proportion of developer's or organization's commits over the entire code base. However, serious gaps remain in the existing reputation estimation tools. First, none of them collect data from the entirety of OSS projects, and even though GitHub contains the bulk of such projects, many important projects are not hosted or mirrored there. Second, none of the existing tools show the impact of developers' work. Google Scholar, for example, does not only show the papers an author wrote but also provides the number of citations for each paper (an indication that the work was used or a measure of its impact). Third, existing tools have limited social network capabilities that do not inform how important or impactful a developer's collaborators are. By addressing these gaps, we can provide a much more comprehensive overview of a developer's technical and social reputation in OSS, ultimately aiding in the trust building process that often precedes the actual collaboration between developers. To realize this, we present the Developer Reputation Estimator (DRE), a tool that provides a more complete overview of a developer's technical and social reputation at its core while addressing the aforementioned gaps by providing the following features:

1. Making measurements based on a developer's contributions to a diverse number of OSS platforms rather than just one.

2. Providing insight into the impact a developer has on other OSS projects and developers.

3. Providing a more comprehensive display of a developer's social network, including important collaborators and a shortest path to celebrities.

DRE utilizes the World of Code (WoC) infrastructure [15] to obtain over 38 million developer identities used in over 1.8 billion commits gathered from over 116 million non-forked repositories. Developers start by searching through these 38 million identities for the ones they used in their commits. The selected and omitted IDs are both recorded and used as training data to help with identity correction. Once a user selects the IDs that belong to them, the tool runs a job in the background to identify all commits made using these IDs and, using the WoC infrastructure, obtains all files modified by these commits, repositories where these commits occur, and blobs created by these commits. Further calculations are then done to construct the social network and measure the impact as described below. This profile consolidates their work (in terms of development languages, commits, projects and files) across platforms and finds their collaborators with their corresponding projects, all displayed in a "developer profile".

In the rest of this paper, we begin by describing the challenges associated with processing the WoC data for backend calculations in Chapter 2. In Chapter 3, a complete walk-through of a user's experience with DRE and the system itself is presented. We discuss several potential use cases, along with limitations of the system, in Chapter 4. Chapter 5 provides an overview of similar systems and Chapter 6 gives an overview of future plans for our application.

Chapter 2

Pre-Processing Challenges

2.1 Data collection

It is not possible to see an individual's global impact if the activities of the individual are scattered across multiple platforms but only one platform is used to measure the developer's work. For example, developers might contribute to a variety of OSS forges, such as GitHub, GitLab, and BitBucket just to name a few. Furthermore, each forge may have its own feature set to support OSS development, including but not limited to version control tools, issue tracking, and merging external contributions (pull requests). In order to showcase all contributions of a particular individual, a full collection of their activity that is merged from all platforms is required. In practice, obtaining such a comprehensive overview of activity for every developer is not possible due to the heterogeneous nature of activity traces across platforms, archived traces that may not be available anymore, and other inconsistencies. However, the World of Code (WoC) infrastructure [15] serves as the best attempt at aggregating all of this data by using a shared feature across the majority of forges. This shared feature is the version control system (VCS) called Git[1], which keeps track of a software project's life cycle from beginning to its present version. At its core, projects under Git version control consist of objects called blobs, trees, and commits. As a brief overview, blobs are essentially each of the files in the project. Trees are the directories that contain a specific subset of file for the project. Lastly, commits consist of both trees

[1] https://git-scm.com/

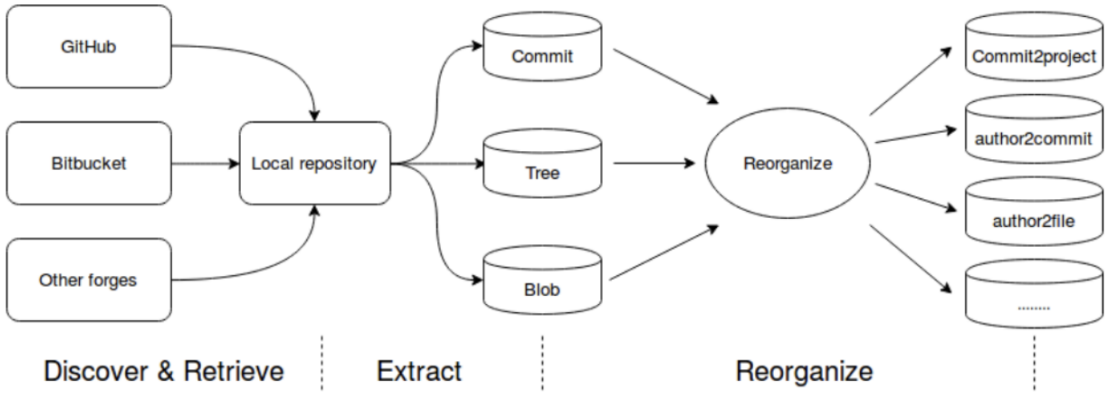

Figure 2.1: The World of Code (WoC) Infrastructure Overview [15]

and blobs at a specific moment in time for the project. Commits also included important metadata, including the author of the commit, the date and time of the commit, and a message describing the commit. As developers work on a project, they create, modify, or delete blobs and trees. When they want to save their work, they make a commit which provides a snapshot of the project at that point in time, allowing them to revert back to it should any issues arise with the project in the future. With Git at the core of data collection, WoC is able to scrape activity traces from all forges with Git support by extracting the blobs, trees, and commits from each project and then associating the commits with their respective author or developer as seen in Figure 2.1. This results in several useful mappings between a developer and their activity traces, including but not limited to a developer and all projects they have worked on, all files they have modified, and many more. We utilize the resulting WoC mappings between developers and their activity traces as the backend for DRE.

2.2 Data quality

While WoC provides mappings between developers and their activity traces, there still exists some data quality challenges before those mappings can be used to their full advantage. Specifically, because DRE considers the relationship between developers and projects in some of its calculations, it is important to carefully measure both of these entities in the

context of the WoC infrastructure. However, both author IDs and projects are susceptible to quality issues, such as multiple author IDs per developer and project cloning.

2.2.1 Author Identities

One of the biggest challenges arises from the fact that a single developer may use a variety author IDs when creating a commit. At its core, WoC extracts the author ID tag attributed with each Git commit. These author IDs are represented by a combination of the authors' names and email addresses, in the following format: `first-name last-name<email-address>`. As an example, a developer whose first name is "John", last name is "Doe", and email address is "john@me.com", the corresponding author id in the WoC dataset would be: "`John Doe <john@me.com>`".

In a perfect world, this author ID would correspond to a single developer, and we could then use that author ID to aggregate all commits associated with the author ID and use it as the basis for DRE. However, this is seldom the case because developers are allowed to free-fill this author ID tag with each commit as they choose. Furthermore, this author ID tag may differ from machine to machine that a developer uses. As a result, a developer might have several additional author IDs (that they might not even be aware of) inside WoC that collectively correspond to a single developer. An example of this is shown in Figure 2.2, which shows three different commits I made on GitHub to a project. The middle row corresponds to a commit I made on my desktop while the two others were made on my laptop. In the WoC dataset, these two identities appear as `Andrey <andrey@utk.edu>` and `zol0 <andrey@utk.edu>` respectively. When DRE performs calculations on my activity traces, it would ideally use *both* identities to extract all of my commits rather than treating each one as a separate developer.

To address this, we use a dataset shared by Fry et al. [8] that resolves the 38 million author identities in WoC (version Q) by creating blocks of potentially related author IDs (e.g. IDs that share the same email, unique first/last name) and then predicting which IDs actually belong to the same developer using a machine learning model. The approach identified over 14 million author IDs belonging to at least one other author ID. From this set, around 5.5 million developers were identified, with a median of two author IDs per developer.

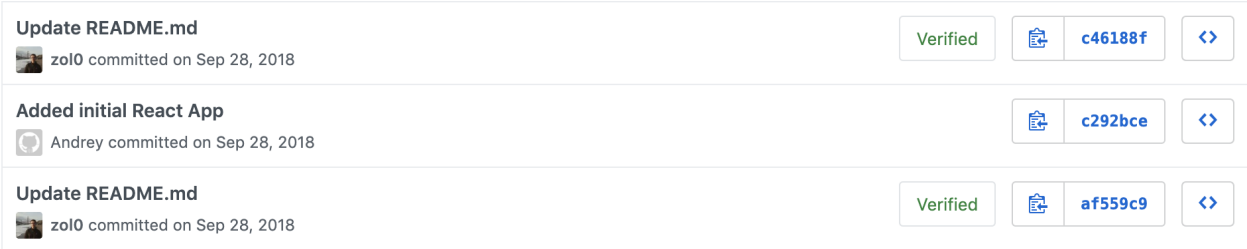

Figure 2.2: Two Identities (Andrey and zol0) for the Same Developer

When calculating the measures described in this paper, we identify each developer using the new associations created by the identity resolution approach. This allows us to create a much more accurate representation of each developer's activity traces and helps us avoid comparing two author IDs that are in fact the same developer.

2.2.2 Duplicate Projects

The inherent ease of cloning or forking Git projects creates another unique data cleaning problem for WoC, which has over 100 million projects. Many of these projects are clones or forks of each other, and these clones often make little to no changes to the parent project, creating a challenge similar to the previous one and posing several problems for our calculations with DRE. One such problem is that a developer who contributes to a highly-cloned project will have their commits appear in the remaining cloned projects as well. Specifically, if a developer contributes to one project using the `Python` programming language and ten other people clone this project and make little to no changes, the developer would be attributed with having worked with `Python` on eleven different projects, rather than just one.

To address this, we use the dataset published in [20]. The authors apply the Louvain community detection algorithm to a massive graph consisting of links between commits and projects in WoC (because two projects are highly unlikely to share the same exact commit unless they are clones). We leverage that work to combine commits from the forked projects. The shared dataset allowed us to ensure that we do not count the same project-related information multiple times due to these forks/clones.

2.3 Data size

As mentioned before, the WoC dataset scrapes Git objects from a variety of forges, and as of its most current version at the time of this paper (version Q, containing all data collected through December 2019), it contains around 7.2 billion blobs, 1.8 billion commits, 7.6 billion trees, 16 million tags, 116 million projects (distinct repositories), and 38 million distinct author IDs. When working with this large of a dataset, certain calculations can take quite a long time (discussed in more detail in Chapter 3.2.6), which is not convenient for time-sensitive operations. For example, to locate the initial set of author IDs for a particular developer, an author database of over 38 million IDs is queried against several search parameters provided by the user. This process is done in real time and users expect results within seconds. Another time consuming operation comes soon after author selection, where a user waits for the backend to perform aggregations and network calculations on graphs of hundreds of millions of nodes and edges. To address these concerns, the databases used to store the data are sharded across multiple machines for higher throughput and faster queries. Furthermore, for the time-sensitive author lookup, proper indexes are placed on the database schema to allow for quicker matching, especially through the use of text indexing. For the expensive backend calculations, an initial set of blacklisted entities (e.g. a Git project that is trying to reach 1 million commits artificially) that mislead graph traversals and kill computation time has been created. Also, proper triggers have been set up to ensure that these calculations are only run when needed. An example of this includes using a "last-updated" date within the database to notify the backend of any changes a user might have had, such as the addition of new author IDs, ultimately resulting in a new set of calculations for the user's dashboard.

Chapter 3

DRE: Walk-through and Design

DRE is presented in the form of a web application where users can sign up to generate their "developer profile". This profile consists of a variety of measures, each of which gives some insight into the developer's technical expertise and social network. Once their profile is generated, a developer can utilize it as a "resume" for their open source contributions, ultimately aiding them in establishing trust with other developers in an efficient manner. The rest of this chapter presents a walk-through on how to use DRE, the backend organization, and then breaks down all the different developer profile measures.

3.1 Usage

3.1.1 Sign-up and Search

To use the tool, a user must first create an account by providing an email address and a password for the account. Once an account is created, the user is then requested for additional information, as seen in Figure 3.1, including the user's first and last names, any additional email addresses, and any user names (i.e. email handles, GitHub user names, etc.). Recall from Chapter 2.2 that commit author IDs are in the form `first-name last-name<email-address>` and that most developers have several author IDs associated with their commits whether they are aware of them or not. Thus, the more information the user provides, the more author IDs we can locate that potentially belong to that user. If the

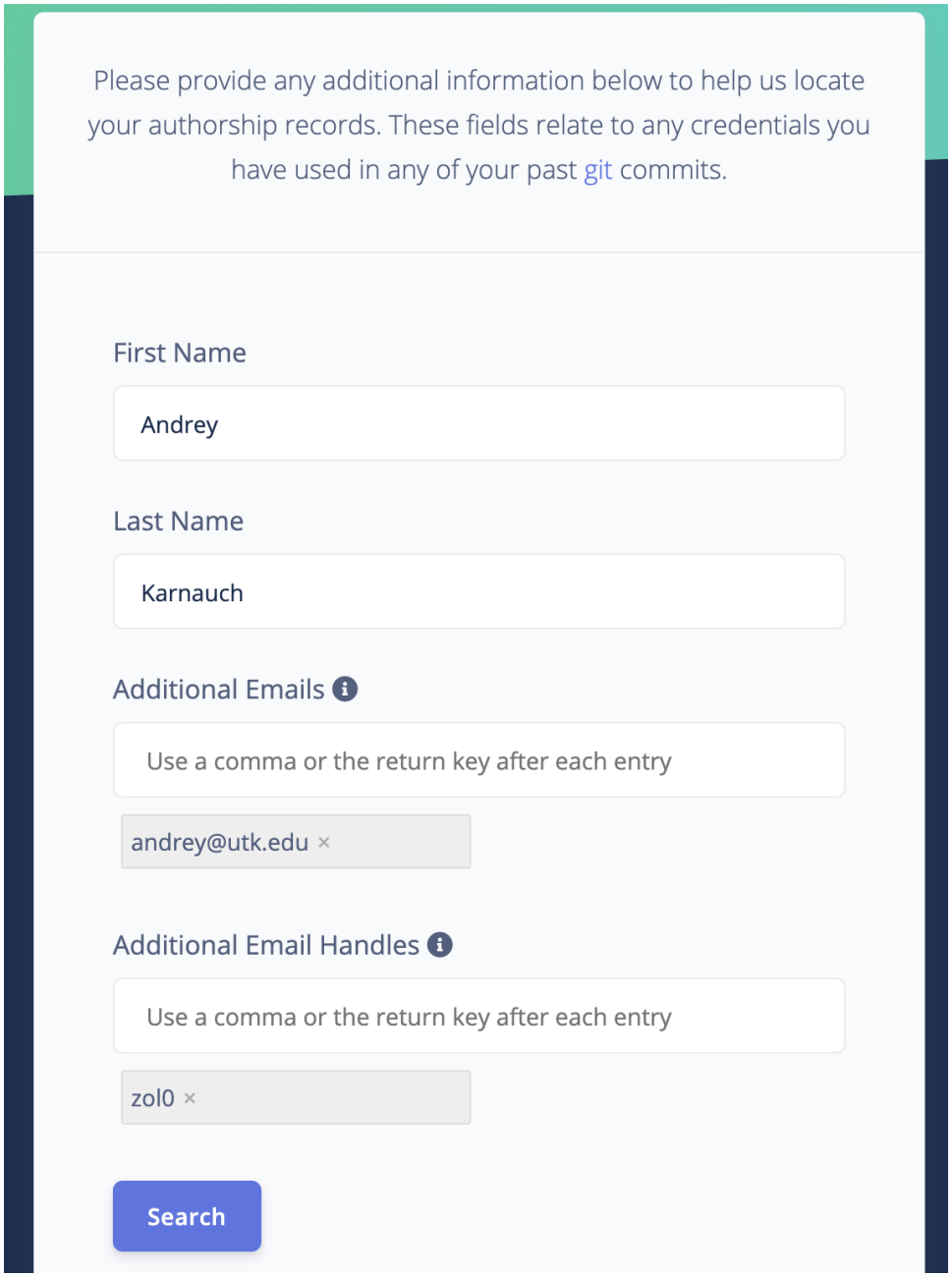

Figure 3.1: The form a user is asked to fill out in order to locate potential author IDs for the user. Sample responses are shown.

user does not provide any additional fields, we simply conduct the search using they email the signed up with.

Once the user submits this form, a search is performed against a MongoDB [1] database containing author IDs in a collection named `Author`. The `Author` schema is built directly from information in WoC and contains the first and last name, email, user name and author ID (`first-name last-name<email-address>`) extracted from a Git commit. Using MongoDB (a NoSQL database) allows for efficient text indexing, which we utilize to match user-provided input against the millions of strings stored in the `Author` collection. Because this is a time-sensitive operation in which a user expects results within a few seconds, we achieve even more speedup by sharding the `Author` collection across four machines, allowing each machine to process only a subset of the original 38 million IDs. Finally, to ensure a timely response in the face of a basic query that returns too many results, we limit the amount of author IDs returned from a user's search to one hundred and provide an error to the user if their query exceeds the cap. This limit is necessary to avoid returning millions of author IDs for generic queries (e.g. providing a first name of "John" and nothing else).

The search returns a list of IDs that are potential matches for the user based on string comparisons of the search parameters provided by the user. The user is then requested to select all IDs that belong to them by simply clicking on whichever author IDs the user recognizes based on the information presented. Selecting an author ID turns it blue and places a checkmark in a box, as seen in Figure 3.2. The last record shown in this figure is also an example of how Git commits allow free-form entry for the author ID (in this case an email is not provided by the user, just a handle).

3.1.2 Backend Processes

After selections are made, the user's data is stored in another MongoDB collection. The `User` collection is populated during account creation and the schema includes the user's email, password hash, search parameters, selected IDs and omitted IDs (all IDs from the search not selected by the user), and the last updated date (determines whether or not a

[1] https://www.mongodb.com/

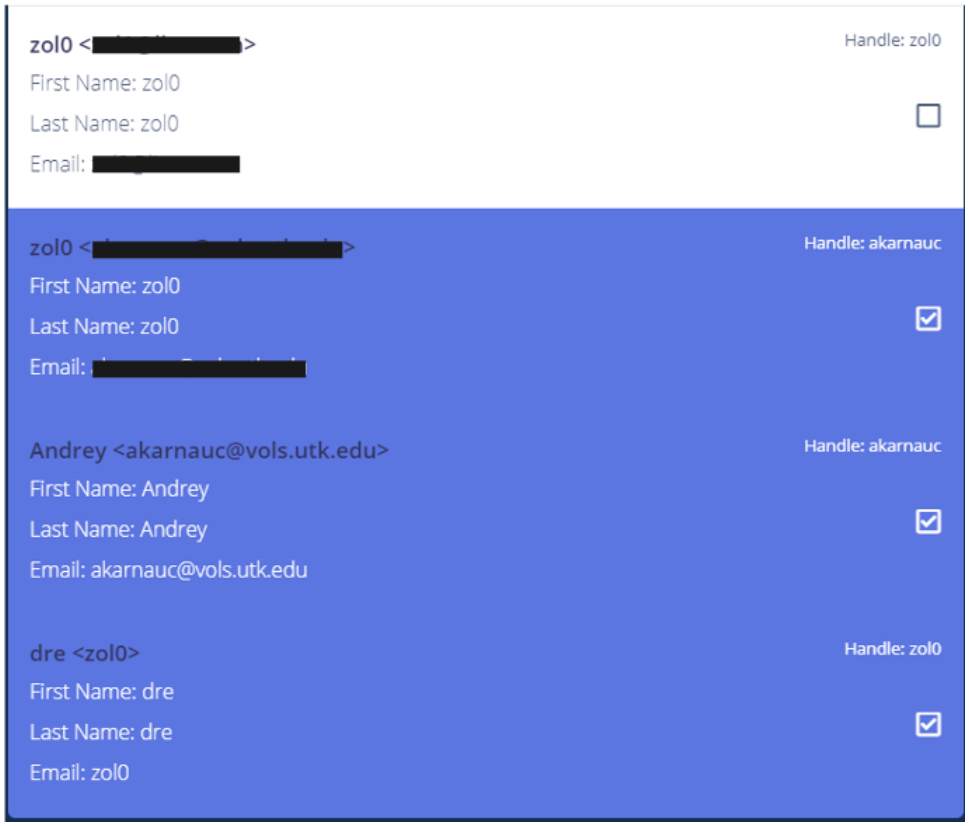

Figure 3.2: The results page of potential author IDs that belong to a developer. The IDs a user recognizes are chosen (blue) and unfamiliar ones are left alone (white).

new set of calculations is needed for this user). The `User` database feeds the selected and omitted IDs directly to the backend for its main operations. The selected and omitted IDs are used by DRE's backend in two ways: (1) A dashboard is populated with a number of measures of the user's activity, a result of the amalgamation of all activities of all selected IDs, as we discuss below and (2) As an input to the disambiguation algorithm that compares all the selected and omitted IDs with the 38 million author IDs in the dataset to predict which IDs belong to a single developer using the machine learning technique described in [8]. While account creation is the main trigger of backend calculations, any update to the user's selected and omitted IDs (e.g. if the user selects or omits additional author IDs that are recommended as a result of the disambiguation algorithm) is also recorded, indicating a need for re-calculation. Once the backend operations finish, the `Profile` database is populated and/or updated with the measures listed below. The tool, a layout of which is depicted in Figure 3.3, notifies the users via email that their developer profile is ready for viewing.

3.1.3 Upload list of IDs

Lastly, researchers in academics and commerce have the option to upload a list of authorship identities from various projects. This list will be fed to the disambiguation process [8] in the backend which can correct low quality developer identity data that matches the author ID format present in the WoC dataset. The details of this are discussed further in Chapter 4.1.2.

3.2 Measures

The following measures are calculated using the WoC mappings that associate author IDs with their respective Git objects. The specific mappings are discussed in more detail as the measures are presented. Furthermore, it is important to note that all of the measures are calculated for each author ID that the developer selected as belonging to them and combined to provide the final results.

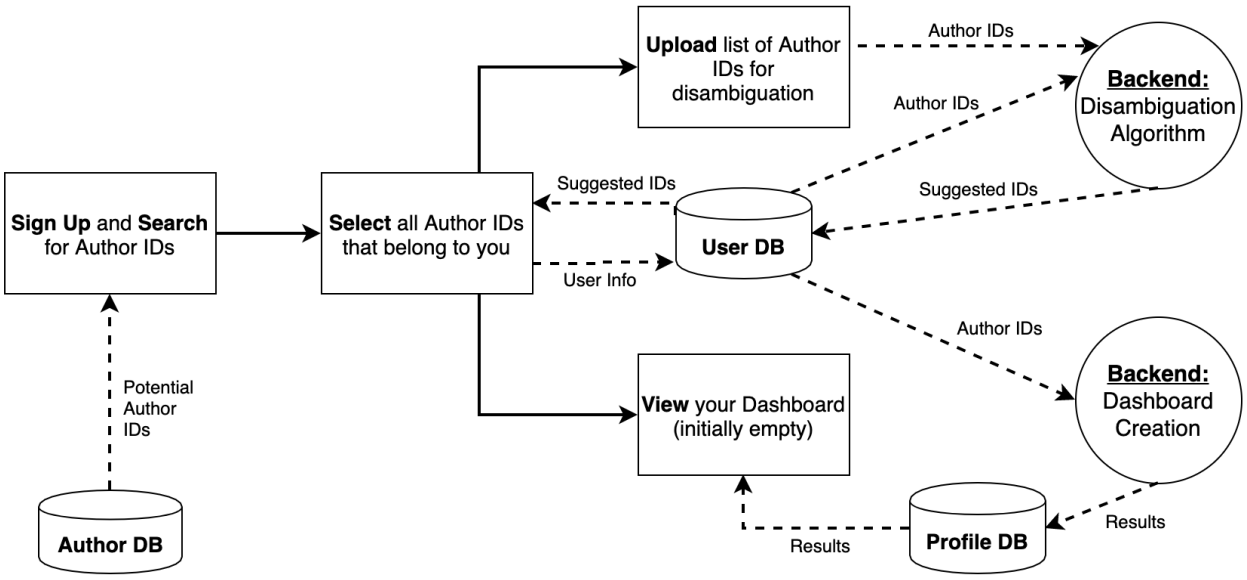

Figure 3.3: The front and back-end flow of DRE (from left to right).

Figure 3.4: The basic aggregation measures displayed at the top of a developer's profile.

3.2.1 Aggregated Contributions

The first measure generated by DRE is displayed at the top of the developer profile and includes the following aggregations:

1. The total number of commits the developer has made. This is obtained from the author-to-commit mapping within WoC.

2. The total number of projects the developer has worked on. This does not include any projects that cannot be scraped (e.g. private repositories on GitHub). The is obtained using the author-to-project mapping within WoC.

3. The total number of collaborators a developer has worked with. As of now, this is a simple count that includes all other contributors to a project that a developer has made a commit to. This measure requires us to first use the author-to-project mapping and then the project-to-author mapping to obtain all the developers who worked on a specific project. These developers across all projects the developer has worked on make up total collaborator count.

4. The total number of files a developer has created or modified. This is obtained from the author-to-commit mapping. Once we obtain all commits for the developer, we utilize the commit-to-file mapping to obtain all the files the developer has created or modified within their commits.

An example of how this information is presented can be seen in Figure 3.4. While these are some of the basic measures that other forges may already provide themselves, they are

important to include due to the insight they provide about a developer's activity. The number of commits help show the overall lifetime activity of a developer. A higher project count can serve as a reflection of the experience, passion and drive of a developer. The collaborator count can reveal a developer's willingness to work with others. The number of files modified can show the size of projects a developer works on. Additionally, when combined with some of the other measures we present on the profile, a better discernment of a developer's activity can be made. For example, a developer with a modified file count of 3,000 is more impressive if their main programming language is C compared to if their main language is JavaScript, because JavaScript projects tend to have far more automatically generated files that may inflate the file count (e.g. the `node_modules` directory in node.js projects).

3.2.2 Projects

The next section of a developer's profile provides a more in-depth look at the projects the developer has contributed to. This list of projects is obtained using the author-to-project mapping and then mapping the projects to their commits. Commits belonging to the developer are then counted alongside the total number of commits. To avoid listing every single project a developer has worked on for more active developers, we limit this list to the top 30 projects in terms of the number of commits a developer has made. The projects' names are shown along with the number of commits the developer made to the project as well as the total number of commits made by all contributors to the project. Furthermore, when available, each project name is hyperlinked to its original forge-specific repository, allowing users to navigate to the project's original source. This part of the profile provides a variety of insights into the kind of projects a developer works on. Firstly, the name of projects can be used by the person viewing the profile to quickly see if they are familiar with any of them. Second, the two separate commit counts allows others to assess the percentage of overall contribution the developer has made to each project, a reflection of how involved the developer was with each project. Lastly, the total number of commits made to each project by all contributors can reveal the size of projects the developer tends to work on. An example of how this list is displayed on the profile is shown in Figure 3.5.

Your Projects

PROJECT NAME	YOUR COMMITS	TOTAL COMMITS
dsande30_COSC402	39	185
dsande30_COSC560-PA1	23	42
dsande30_COSC560-PA2	19	28
zol0_PA2	18	21
fdac18_FlightCost	14	47

Figure 3.5: A list of project's the developer has worked on along with their commit count vs. the total commit count.

3.2.3 Programming Languages

DRE also measures a developer's programming language breakdown by looking at the file extensions associated with each of the developer's commits. The WoC mappings currently include C, C#, FORTRAN, Go, JavaScript, Python, R, Rust, Scala, Java, Perl, Ruby, and Julia languages, as well as source code present in Jupyter Notebooks [2]. This measure is presented in the form of a pie chart, as seen in Figure 3.6, revealing the coding languages used by the developer along with the number of files associated with each language. This ultimately shows the user's language preference and potentially expertise. However, as mentioned earlier, this measure is susceptible to inflation due to auto-generated files in certain languages, such as JavaScript.

3.2.4 Collaborators

In addition to the collaborator count presented earlier, DRE also provides more insight into a developer's top 30 collaborators. Clicking on any of the collaborators in the list provides an overview of all the projects the developer and collaborator have worked on together. Furthermore, the collaborator's commit count (a measure of contribution) is listed for each project along with the total number of contributors to that project. This demonstrates the productivity of the collaborator on each project as well as the size of the project in terms of the number of contributors. An example of how this measure appears on a developer's profile is shown in Figure 3.7.

3.2.5 Torvalds Index

Like Erdös number[3], the Torvalds index is the shortest path, with collaborators as nodes and projects as edges (seen on mouse-over), to Linus Torvalds, the creator and developer of the Linux kernel. This measure was originally calculated on-demand by utilizing several different WoC mappings to generate a graph with author IDs as nodes and the projects they share as edges and then running a shortest path finding algorithm between the developer

[2] https://jupyter.org/
[3] https://en.wikipedia.org/wiki/Erdos_number

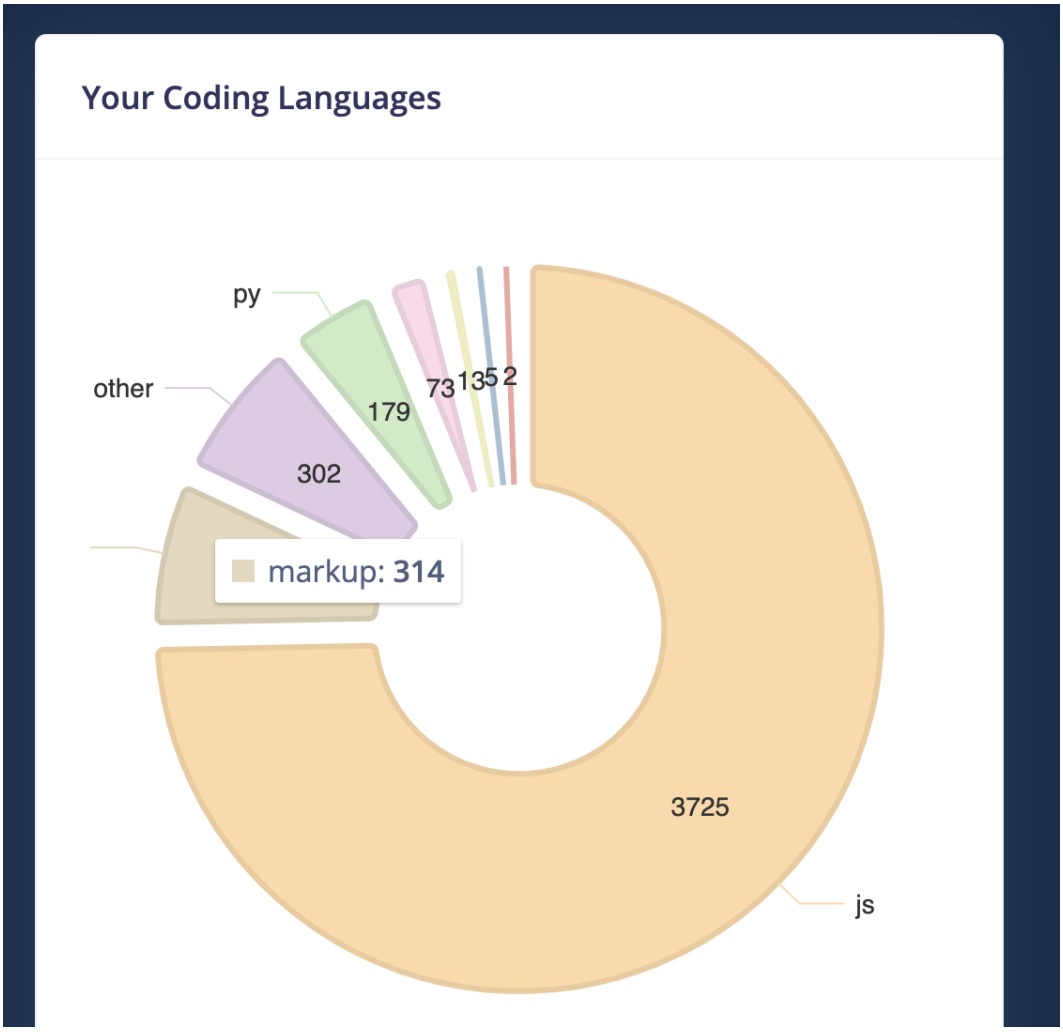

Figure 3.6: A pie chart that breaks down the different programming language usage of a developer.

Audris Mockus <audris@utk.edu>

PROJECT NAME	THEIR COMMITS	TOTAL PROJECT COLLABORATORS
ssc-oscar_gather	47	1
fdac18_Miniproject3	35	45
fdac18_MiniProject2	28	54
EvEng_students	21	20
fdac18_Practice0	12	35

Close

Figure 3.7: A breakdown of the projects a developer and collaborator (Audris Mockus in this case) have worked on.

and Linus Torvalds. We manually created a set of author IDs that we believe belong to Linus Torvalds based on public information. However, this is quite an expensive calculation to do on-demand each time. As a result, we calculated the Torvalds Index for every author ID in the WoC dataset and created a direct mapping, bringing the computation time down to just a single database lookup. Apart from being cute, this index allows a developer to measure his or her position in the social network graph defined by people they collaborate with. It can provide further insight about a user's collaborators, revealing projects or persons that may be of interest to the user, as seen in Figure 3.8.

3.2.6 Blob Duplication

Google Scholar Profiles show the impact of an academic through citation count. Similarly, we measure an aspect of the impact of a developer through the number of reuses of their code. Specifically, we identify blobs of which a developer is the original creator, and then count the number of commits by others containing the developer's blob, an indication that others used the developer's exact file in their own projects. To achieve this, we first iterate through each blob created by a developer's commit and determine an original author of each blob. To obtain this author, all commits creating the blob are gathered and the first commit (in calendar time) is identified. The creator of that commit is considered to be the creator of the blob. In the impact measure, only blobs where the developer is the creator (as defined above) are considered. For each such blob we count the number of commits done by other developers. Such commits indicate that these other developers have copied the original author's code for use in their projects: similar to academic citations indicating the use of others' results. The user is shown a list of blobs which they have created along with a number of commits done by others creating the same blob and the authors of these commits. Listing the authors of these blobs allows the user to see their impact in the social network graph by observing which of their collaborators (if any) used their code/blob. This is the most expensive calculation that DRE performs since blob (file) count tends to be far greater than commit or project count, and we have to iterate through all of said blobs for not only the developer whose profile is being generated, but for all other authors who have the same blob in any of their commits. Furthermore, while this blob duplication count

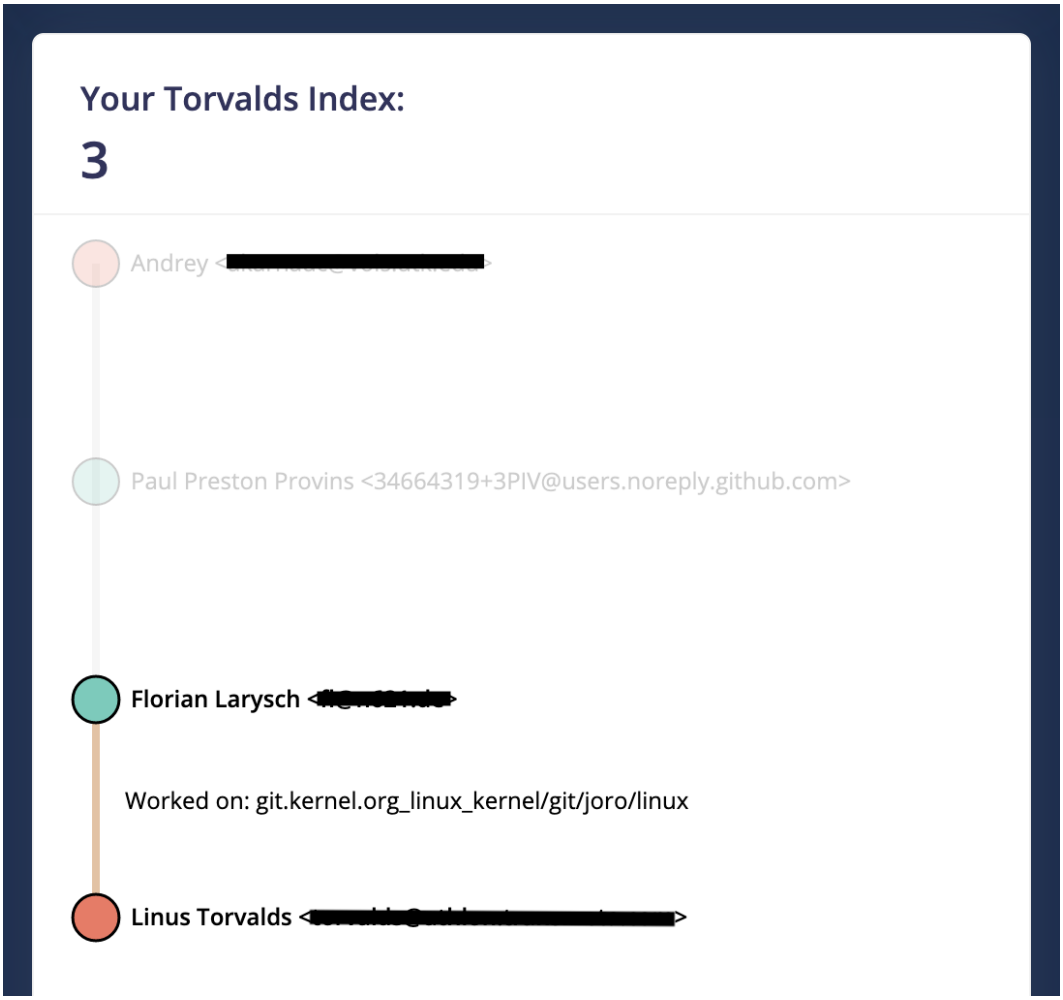

Figure 3.8: Torvalds Index: The path between a developer and Linus Torvalds, with author IDs as nodes and projects as edges.

Your Blobs

BLOB SHA	DUPLICATIONS	CHILDCOMMITS	USERS
9ef92d71275ab49cb118c2c412c24ff16ee7c1f6	6	27	{"b355b
7646888430962463d1e0159d5f36d36c8ca6e5dd	4	52	{"00c986
7cc1735d290958713e748face77be0f9cde88390	4	40	{"67345a
5149d9460ce45ee5e5316b563a62c3f4d66898c5	4	16	{"151f34
4d010d4e7fcc5e3ba75e4df7d8f111ed311412b5	4	13	{"e33f3e

Figure 3.9: Blob Duplication Counts

serves as one of the more important measures in DRE, it remains a challenge as to how this information should be displayed to the user. We use a table as seen in Figure 3.9 with the SHA1 hash of the blob as generated by Git and its duplication count as described above. Furthermore, we provide the number of child commits for the blob, which is the number of commits made after the commit that introduced the blob. This allows the user to gauge when the blob was introduced in the project life cycle, with a higher child commit count indicating early introduction of the blob and a lower commit count indicating the opposite. Lastly, we provide a list of `commmit:collaborator author ID` pairs to show which of a developer's collaborators used their code along with the commit they used it in as described earlier.

3.3 Software Architecture

In order to bridge the backend calculations with a visually appealing frontend, we built DRE using the MERN software stack. Specifically, this includes the following technologies:

MongoDB [4] (M) for database management, Express JS [5] (E) as the web framework, React JS [6] (R) for the frontend, and Node JS [7] (N) as the server for our web application. We chose to use this software stack for several reasons. First, it utilizes some of the most popular technologies in modern web development, allowing us to stay on par with other applications. Second, all levels of the stack use the JavaScript programming language, making development more efficient once familiarized with JavaScript. Third, with the popularity of each technology in the stack, there exists an abundant amount of documentation for development, allowing any new developers for DRE to catch up on the internals in an efficient manner. However, there are also some drawbacks to this stack, as the technologies are still being developed. Specifically, with React JS, we have noticed a complete overhaul of its internals lately, requiring us to re-write most of the React JS code that communicates with the backend because many of the functions we originally used have now been removed or replaced. However, React JS does provide extensive debugging messages that guide the developer through the necessary steps needed to convert older code into the new standards.

While we presented an overview of the front and backend flow of DRE in Figure 3.3, we now describe it in the context of the MERN stack that DRE uses. Specifically, as seen in Figure 3.10, we begin with a process named `watch.py`, which periodically scans the `User` collection in the database for any new users or any new changes to a user's set of author IDs. If a change is detected, this process calls another process, `sdpg.perl`, which performs all of the necessary calculations to obtain each of the measures we laid out in the previous section. Once `sdpg.perl` has finished its calculations, it stores all of the user's measures in the `Profile` collection. At the same time, `watch.py` sends out an email to the user notifying them that their profile has either been created or updated. Now, the MERN stack comes into play to help visualize all of these measures. At this point, the Node JS server is running and our front-end is being served using React JS. Once a user visits their profile, the React JS frontend triggers a request to the extract the user's measures from the `Profile` collection. The request is passed down through the Express web framework, where it eventually reaches

[4] https://www.mongodb.com/
[5] https://expressjs.com/
[6] https://reactjs.org/
[7] https://nodejs.org/en/

a separate library called Mongoose [8]. Because MongoDB is a NoSQL database and can have records with arbitrary fields, we enforce a schema that we expect to retrieve from the database using Mongoose. This schema is built directly from the measures that `sdpg.perl` produces and stores in the database. Once these measures are retrieved, they are sent back to the React JS frontend using routing provided by Express JS. From there, React JS caches the measures locally and utilizes its frontend capabilities to display the data back to the user.

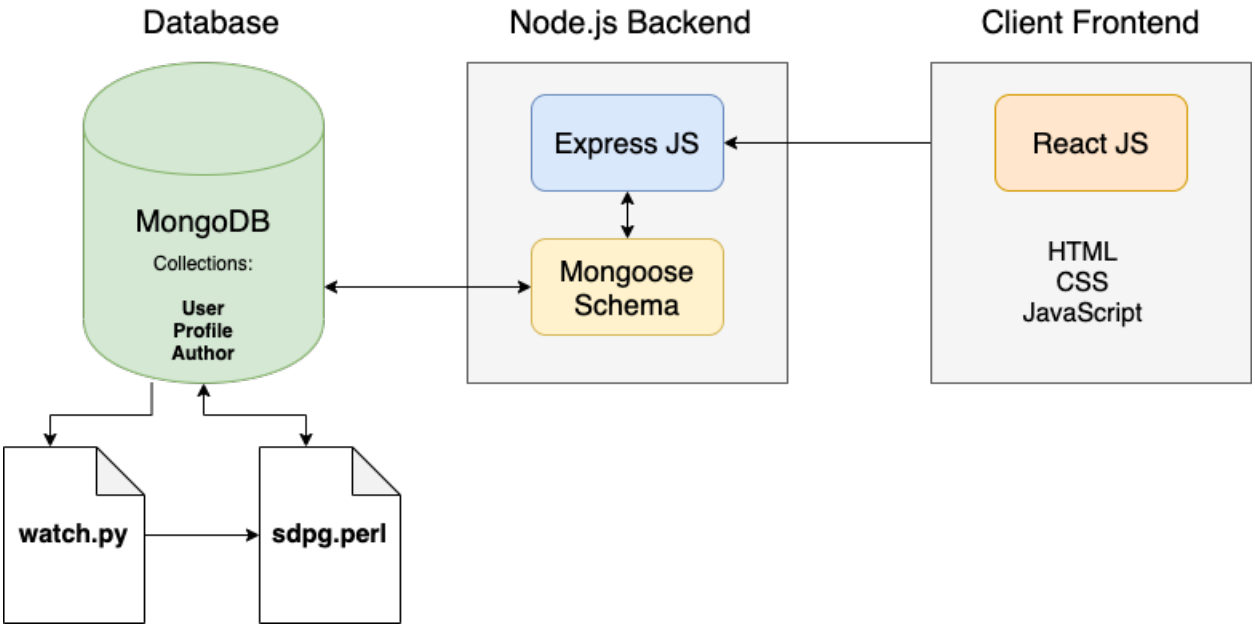

Figure 3.10: The MERN stack as used by DRE

[8] https://mongoosejs.com/

Chapter 4

Use Cases and Limitations

4.1 Intended Users

DRE is designed and built keeping a number of users and challenges in mind. For example, a service that consolidates developer activity, much like scholarly activity showcased on Google Scholar Profile or work history as on LinkedIn, is not available for software engineers. Measures such as the researchers total citation count, the h-index etc. showcase the level of expertise of an academic scholar in a holistic sense, and a more granular measure of citation count for a given scientific publication showcases the impact of the researcher in a particular domain. Similarly, a number of measures such as a developer's commit count, the number of projects the developer has worked on, the languages the developer uses for coding and the number of collaborators can indicate a developer's expertise in a domain. Furthermore, a developer's impact can be measured by calculating the extent of reuse of the code created by the developer. Here, we outline the potential users of DRE and the benefits it can bring them.

4.1.1 Individual Developers

DRE aggregates a developer's contribution from multiple forges (and from multiple author IDs found in Git commits) and creates a consolidated measure of impact and enables showcasing their areas of expertise in the form of a personalized profile. The profiles of

users can be helpful in assessing their experience, impact, and contribution to OSS. From the user's standpoint, this tool can provide evidence to support their stature of professional expertise as well. Summarized data can be used as a signal for the quality of the work they are doing and, therefore, increase the confidence that their issues and/or pull requests are of high quality, ultimately helping them get accepted. DRE enables developers to also simply quench their curiosity regarding their own contributions, impact, social centrality, etc.

4.1.2 Researchers

The software engineering community is going through an evolution of practices with contributors adopting social platforms for development and knowledge exchange. As a result, researchers in commerce and academia are trying to understand the participation and contributions of developers in this collaborative environment. Mining software repositories help researchers answer a number of questions, such as "Who are the most influential developers?", "What is a developer's contribution across platforms?"[2], "What are the roles of the various contributors?"[18]. Unearthing answers to these questions help researchers understand the OSS development process better in terms of realizing who's driving the evolution process and who are the central people to a software project. To find answers to these, a complete set of data containing software development activities across all platforms is required as realized in, for example [15]. For any research to be effective, the data also needs to be free of errors, especially errors related to misrepresentations of the the developer identity. DRE allows researchers to upload and correct a list of developer IDs through advanced methods of disambiguation carried out in the backend and validated by the developers themselves.

4.1.3 Recruiters

Seeking expertise required for a particular software development task is a challenging process. While job seekers may claim proficiency in various languages and domains, verification of the claim requires large amounts of resources such as time (for lengthy interviews) and expert judges of performance. An online profile of developers can act as a bridge between

job seekers and potential employers by corroborating claims of expertise. Data from OSS repositories have been useful for discovering technical expertise in the past [24]. For example, a developer's contribution to a project in terms of commits and number of commits made in various languages can corroborate claims made in a resume by a job seeker. DRE reserves the capacity to introduce querying capabilities metrics in the future that can provide further assistance to recruiters.

4.2 Initial Feedback

Before reviewing some of the limitations of DRE, it is worth while to discuss the evaluation of DRE by other users. We have yet to conduct a formal user study with DRE because we feel it is still missing enhanced support for the aforementioned use cases (e.g. allowing functionality to search for developers based on certain criteria present in the profiles) and some major features that would aid the usability of DRE as well as, which we discuss further in Chapter 6.1. However, we have encouraged over 40 colleagues and peers to generate their developer profile using DRE and received some feedback that helps fuel the ideas presented in Chapter 6.1. Their aid has also helped us realize some of the limitations of DRE as well.

4.3 Limitations

Some of the main concerns with DRE revolve around artificial inflation of DRE's measures, such as generating thousands of throw-away commits, project, etc. While this can be done to inflate the basic count measures, DRE still provides a transparent look at the developer's projects and collaborators. As a result, those viewing a developer's profile with large, artificially generated counts are able to see the project(s) that the commits belong to and even visit the project's source. This transparency allows others to quickly determine if the developer is working on genuine projects or intentionally boosting their measures. Some other limitations of DRE arise as a result of how we calculate some of our measures. Two main examples of this are the collaborator counts and programming language breakdown. For the total number of collaborators calculation, we have received feedback that this count is

often much higher than the user expected because it counts all other contributors to projects you have worked on, which does not always imply that you collaborated with them directly. We plan on making this measure stricter in the future, by only counting collaborators as those who have modified the same files as you. For the programming language breakdown, we feel that it provides a good overview of what languages a developer has worked with. However, it does not provide much insight into the developer's domain-specific expertise, which motivates some of the future work we discuss in Chapter 6.1 that deals with API extraction.

Chapter 5

Related Work

DRE is not the first attempt at measuring developer experience, and in fact, this interest in a developer's expertise has been around since the beginnings of software development [3, 1, 6, 5]. The focus at the time, however, was quite different. For example, much of the early work was motivated by the need for estimating project costs and measuring the size of software projects independent of the language the project was written in [4]. As this work made ground, researchers began to shift their focus on the developer-project experience, where they found that developers expend a lot of effort to fully understand the internal structure of new software projects they join [26]. From this, approaches such as Expertise Browser focused on developer expertise within a specific project [19]. They measured this expertise based on the modifications the developer made to source code files within the project under the assumption that any modification required a specific set of knowledge and understanding about both the project and files modified. However, as the OSS community grew, interest shifted from project-specific expertise to a more general sense of developer experience that they could bring to any project they joined.

Presently, social coding platforms such as GitHub provide a variety of charts and indicators of developer activity (the timeline of commits) and their social status (followers). This has sparked a variety of research into how developer traces and developer profiles can provide insight about a developer's expertise or reputation. These studies include qualitative approaches, such as the one by Marlow et. al. [17], who showed that your developer profile on GitHub can help other developers gauge your general coding ability and project-relevant

skills, but only at a more general level. Similarly, Singer et. al. [21] interviewed developers and employers to observe how they utilize developer profiles to gauge the quality of a potential new hire. The results showed that profile sites with a "skills" word-cloud representing the technologies (languages, frameworks, etc.) a developer claimed to be familiar with proved to be the most helpful assessment of a developer's expertise. We use these works to motivate some of the measures included in DRE, such as programming language overview and project lists, and also to motivate more specific measures in the future, such as language-specific technologies and frameworks, to help others gauge both overall experience of a developer as well as domain-specific expertise.

There have also been several tools created to help automate the process of identifying developer expertise through social coding platforms. For example, CVExplorer [10] is a tool created to expose developer expertise using a word-cloud of all relevant technologies, frameworks, and general skills by parsing developer commit messages and README files. SCSMiner [25] is another tool created to help identify experts on GitHub based on an arbitrary input query. The authors also obtain expertise attributes by parsing README files of projects a developer has contributed to, but they extend this by creating a generative probabilistic expert ranking model to rank developers based on certain skills or expertise one might be looking for. Lastly, Hauff et. al. [13] attempt to match developers with job advertisements based on a developer's expertise. Their approach involves extracting relevant terms from README files and mapping them to the same vector space as job advertisements. From there, they are able to rank all developer profiles based on the cosine similarity they share with the job advertisements. While these approaches do not cover developer reputation under a large umbrella the way DRE does, they provide several specific avenues of future work for DRE, such as parsing the actual content of files a developer modifies, commit messages, and more and turning those into measures that can be displayed on a developer's profile.

Chapter 6

Conclusions

In this work we present the first attempt at measuring the technical and social aspects of a developer's reputation in the OSS ecosystem using the WoC dataset. We implement many of the useful measures, such as overall activity counts, already deployed on some major social coding platforms in the context of WoC, which scrapes developer activity across all major forges, rather than just one. Furthermore, we create novel measures, such as the Torvalds Index and blob duplication counts to provide more insight about a developer's social network and their impact on others. We realize all of this in the form of a web application named DRE where developers can sign up and have their own developer profile generated, which contains the all the aforementioned measures. With this platform, we also provide an avenue to aid the research on identity correction by creating a dataset of author IDs that we know belong to a single developer and which ones do not. This ground-truth of author IDs can aid researchers in correcting identity errors in the context of Git authorship records and has already been used in approaches such as the one described in [8]. We also receive informal feedback from over 40 peers to steer the prototype of DRE into a more usable application that can undergo a proper user study. Ultimately, this work is a step towards efficiently establishing trust between developers in the OSS ecosystem and overcoming barriers of entry for new developers [22] by making developer experience transparent to all parties.

6.1 Future Work

We have already identified several areas of improvement for the application we have created, both from feedback of peers and colleagues as well as our own realizations. Most of these areas are discussed in more detail in Chapter 4.3. Two of the major usability concerns revolve around exposing developer profiles to others and querying against all profiles. To address these concerns, we plan to allow for an option that makes a developer's profile public if they wish to share it with others. This has already been implemented for users registered on the application, allowing them to view each other's profiles. Furthermore, to enable the use case of recruiters or employers looking to hire, we plan to implement querying functionality, allowing others to search for developers that match a certain set of criteria they are seeking. With this enabled, we can perform a proper user study with the goal of bringing competitors, such as GitHub and LinkedIn, into the picture to compare how our tool holds up against the competition. The study would revolve around providing users with a "task" (e.g. find a developer to work on someone's Javascript project) and evaluating which platform provides the "best" suggestion for each, individual user. From there, we can gauge what our tool lacks in terms of developer profile measures, what users look for when comparing developers, and more.

Lastly, motivated by some of the research discussed in Chapter 5, we have made progress towards exposing even more meaningful relationships between developers, projects, and the domain-specific APIs they use. We believe this can aid our application by providing a recommendation system for developers based on their domain-specific expertise, allowing DRE to recommend other developers, projects, or APIs that might be of interest for the user.

Bibliography

[1] Albrecht, A. J. and Gaffney, J. R. (1983). Software function, source lines of code, and development effort prediction: a software science validation. *IEEE Trans. on Software Engineering*, 9(6):638–648. 30

[2] Badashian, A. S., Esteki, A., Gholipour, A., Hindle, A., and Stroulia, E. (2014). Involvement, contribution and influence in github and stack overflow. In *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*, CASCON '14, page 19–33, USA. IBM Corp. 27

[3] Basili, V. and Reiter, R. (1979). An investigation of human factors in software development. *IEEE Computer*, 12(12):21–38. 30

[4] Behrens, C. (1983). Measuring the productivity of computer systems development activities with function points. *IEEE Transactions on Software Engineering*, SE-9(6):648–652. 30

[5] Boehm, B. (1981). *Software Engineering Economics*. Prentice-Hall. 30

[6] Curtis, B. (1981). Substantiating programmer variability. In *Proceedings of the IEEE 69*. 30

[7] Dey, T. and Mockus, A. (2020). Which pull requests get accepted and why? a study of popular npm packages. 1

[8] Fry, T., Dey, T., Karnauch, A., and Mockus, A. (2020). A dataset and an approach for identity resolution of 38 million author ids extracted from 2b git commits. In *IEEE Working Conference on Mining Software Repositories: Data Showcase*. iv, 6, 13, 32

[9] Gousios, G., Pinzger, M., and Deursen, A. v. (2014). An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355. 1

[10] Greene, G. J. and Fischer, B. (2016). Cvexplorer: Identifying candidate developers by mining and exploring their open source contributions. In *Proceedings of the 31st*

IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, page 804–809, New York, NY, USA. Association for Computing Machinery. 31

[11] Guo, P. J., Zimmermann, T., Nagappan, N., and Murphy, B. (2010). Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 495–504. 1

[12] Hahn, J., Moon, J. Y., and Zhang, C. (2008). Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties. *Information Systems Research*, 19(3):369–391. 1

[13] Hauff, C. and Gousios, G. (2015). Matching github developer profiles to job advertisements. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, page 362–366. IEEE Press. 31

[14] Karnauch, A., Amreen, S., and Mockus, A. (2019). Developer reputation estimator (dre). In *ASE'19*. iv

[15] Ma, Y., Bogart, C., Amreen, S., Zaretzki, R., and Mockus, A. (2019). World of code: An infrastructure for mining the universe of open source vcs data. In *IEEE Working Conference on Mining Software Repositories*. viii, 3, 4, 5, 27

[16] Marlow, J. and Dabbish, L. (2013). Activity traces and signals in software developer recruitment and hiring. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, page 145–156, New York, NY, USA. Association for Computing Machinery. 1

[17] Marlow, J., Dabbish, L., and Herbsleb, J. (2013). Impression formation in online peer production: Activity traces and personal profiles in github. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, page 117–128, New York, NY, USA. Association for Computing Machinery. 30

[18] Milewicz, R., Pinto, G., and Rodeghero, P. (2019). Characterizing the roles of contributors in open-source scientific software projects. In *Proceedings of the 16th*

International Conference on Mining Software Repositories, MSR '19, page 421–432. IEEE Press. 27

[19] Mockus, A. and Herbsleb, J. (2002). Expertise browser: A quantitative approach to identifying expertise. In *2002 International Conference on Software Engineering*, pages 503–512, Orlando, Florida. ACM Press. 2, 30

[20] Mockus, A., Spinellis, D., Kotti, Z., and Dusing, G. J. (2020). A complete set of related git repositories identified via community detection approaches based on shared commits. In *IEEE Working Conference on Mining Software Repositories: Data Showcase*. 7

[21] Singer, L., Figueira Filho, F., Cleary, B., Treude, C., Storey, M.-A., and Schneider, K. (2013). Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, page 103–116, New York, NY, USA. Association for Computing Machinery. 31

[22] Steinmacher, I., Gerosa, M. A., Conte, T., and Redmiles, D. (2018). Overcoming social barriers when contributing to open source software projects. *Computer Supported Cooperative Work (CSCW)*. 32

[23] Tsay, J., Dabbish, L., and Herbsleb, J. (2014). Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 356–366, New York, NY, USA. Association for Computing Machinery. 1

[24] Venkataramani, R., Gupta, A., Asadullah, A., Muddu, B., and Bhat, V. (2013). Discovery of technical expertise from open source code repositories. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, page 97–98, New York, NY, USA. Association for Computing Machinery. 28

[25] Wan, Y., Chen, L., Xu, G., Zhao, Z., Tang, J., and Wu, J. (2018). Scsminer: mining social coding sites for software developer recommendation with relevance propagation. *World Wide Web*, 21(6):1523–1543. 31

[26] Zhou, M. and Mockus, A. (2010). Developer fluency: Achieving true mastery in software projects. In *ACM SIGSOFT / FSE*, pages 137–146, Santa Fe, New Mexico. 30

[27] Zhu, J., Zhou, M., and Mockus, A. (2016). Effectiveness of code contribution: From patch-based to pull-request-based tools. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 871–882. ACM. 1

Vita

Andrey Karnauch was born in Binghamton, NY and moved to Tennessee when he was ten years old. He attended the University of Tennessee for his undergraduate studies, where he was a Min H. Kao Scholar and graduated at the top of his class in 2018 with a bachelor's degree in Computer Science and a minor in Cybersecurity. He remained at the University of Tennessee to pursue his Masters degree in Computer Science. Throughout his undergraduate studies, Andrey held a variety of internship positions, from the Volkswagen plant in Chattanooga, TN to working at the headquarters of the non-profit Engineering Honor Society, Tau Beta Pi, which is located on the University of Tennessee's campus. He was also a member of some student organizations, such as the computer security organization HackUTK, where he held a teaching position for his final semester of undergraduate studies. In his spare time, Andrey enjoys backpacking across various national parks across the United States. He has gone on trips to Yosemite, the Grand Tetons, and Sequioa National Park and hopes to expand this list over the coming years.