

UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Thaís Oliveira Mombach

A Comparative Study of APIs for Querying GitHub Data

Belo Horizonte

2019

Thaís Oliveira Mombach

A Comparative Study of APIs for Querying GitHub Data

Versão Final

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador(a): Marco Tulio de Oliveira Valente

Belo Horizonte

2019

Thaís Oliveira Mombach

A Comparative Study of APIs for Querying GitHub Data

Final Version

Thesis presented to the Graduate Program in Computer Science of the Universidade Federal de Minas Gerais Departamento de Ciência da Computação. in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Marco Tulio de Oliveira Valente

Belo Horizonte

2019

© 2019, Thaís Oliveira Mombach.
Todos os direitos reservados.

Mombach, Thaís Oliveira

M732c A Comparative Study of APIs for Querying GitHub
Data / Thaís Oliveira Mombach. — Belo Horizonte,
2019

xiii, 81 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de
Minas Gerais Departamento de Ciência da
Computação.

Orientador(a): Marco Tulio de Oliveira Valente

1. Computação — Teses. 2. Engenharia de software
— Teses. 3. Software gratuito — Teses. I. Orientador.
II. Título.

CDU 519.6*32(043)

Ficha catalográfica elaborada pela bibliotecária Irénquer Vismeg
Lucas Cruz - CRB 6ª Região nº 819



UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

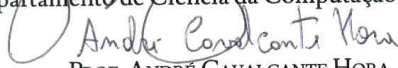
FOLHA DE APROVAÇÃO


A Comparative Study of APIs for Querying GitHub Data

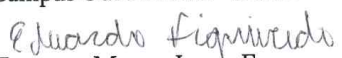
THAÍS OLIVEIRA MOMBACH

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:


PROF. MARCO TÚLIO DE OLIVEIRA VALENTE - Orientador
Departamento de Ciência da Computação - UFMG


PROF. ANDRÉ CAVALCANTE HORA
Departamento de Ciência da Computação - UFMG


PROF^a. LUCIANA LOURDES SILVA
Campus Ouro Branco - IFMG


PROF. EDUARDO MAGNO LAGES FIGUEIREDO
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 3 de Outubro de 2019.

To Paulo, Edileusa, and Aline.

Acknowledgments

Primeiro de tudo, gostaria de agradecer a Deus que me abençoou e deu forças para que eu conseguisse chegar até aqui.

Aos meus pais, Paulo e Edileusa, agradeço por sempre acreditarem em mim e me apoiarem em todas as minhas escolhas, fazendo dos meus sonhos os deles também. Sem todo o carinho e apoio de vocês eu não conseguiria chegar até aqui. Obrigada!

A minha irmã, Aline, agradeço por todo companheirismo e carinho que sempre teve comigo. Obrigada por me ouvir nos momentos difíceis, sem você a caminhada seria muito mais difícil!

Ao Vinícius, agradeço pelo amor e carinho, sempre cuidando de mim e me alegrando meus dias.

Ao meu orientador, Marco Tulio, agradeço por ter me recebido e por todo o ensinamento que me passou.

Aos meus amigos, por me ouvirem nos momentos difíceis e nos momentos felizes comemorarem comigo.

Aos meus colegas da UFMG, agradeço pela ensinamentos e conselhos durante esses anos de mestrado.

Ao meus colegas da SAP, agradeço pela força que me deram durante esse período do mestrado.

*“As we go through life, you will see that there is so much that we don’t understand.
And the only thing we know is things don’t always go the way we plan.”*

(The Lion King)

Resumo

O GitHub é uma plataforma usada por 40 milhões de usuários para hospedar código-fonte de mais de 100 milhões de repositórios. A plataforma é usada também por importantes sistemas de código aberto (OSS) para armazenar e compartilhar seu código, e também para organizar as contribuições de desenvolvedores de todo o mundo. Esse fato, juntamente com o fácil acesso aos dados, explica por que o GitHub é amplamente usado em pesquisas de Engenharia de Software. Esta dissertação de mestrado visa fornecer uma análise comparativa e detalhada de duas APIs populares para acessar dados do GitHub: API REST do GitHub e GHTorrent. Primeiro, realizamos um estudo usando 23 consultas extraídas de artigos publicados em duas grandes conferências de Engenharia de Software. Implementamos essas consultas para analisar as APIs em diferentes dimensões, incluindo esforço de configuração, qualidade da documentação, informações disponíveis, complexidade das consultas e limitações. Com base nessas análises, foi possível comparar as duas APIs e descobrir que a API REST do GitHub pode estar à frente do GHTorrent nos aspectos analisados. Por exemplo, ambas as APIs fornecem documentação, mas a documentação do GHTorrent parece estar desatualizada. Finalmente, concluímos o trabalho principal avaliando as duas APIs em um estudo de caso. Coletamos os 10K principais repositórios com base no número de estrelas para responder a três perguntas de pesquisa sobre desenvolvimento de software de código aberto em diferentes países. Com este estudo, foi possível analisar os resultados recuperados usando dados de diferentes APIs. Identificamos algumas variações nos resultados, a maioria relacionada à diferença na data de coleta dos dados das duas APIs. Como contribuições práticas do trabalho, são apresentadas análises de cada uma das APIs para auxiliar os pesquisadores na escolha de uma API para coleta de dados.

Palavras-chave: GitHub, API REST do GitHub, GHTorrent, desenvolvimento de software aberto.

Abstract

GitHub is a popular platform for hosting source code with more than 100 million repositories and 40 million users. GitHub is used by important Open Source Software (OSS) projects to store and share their code, and also to organize the contributions from developers around the world. This fact along with easy access to GitHub data explains why GitHub is widely used in Software Engineering studies. This master dissertation aims to provide a detailed comparative analysis of two popular APIs for accessing GitHub data: GitHub REST API and GHTorrent. First, we conduct a study using 23 queries extracted from papers of two major Software Engineering conferences. We implemented these queries to analyze the APIs under different dimensions, including setup effort, documentation quality, data coverage, queries complexity, and limitations. Based on these analyses we could find that GitHub REST API is ahead GHTorrent in the analyzed aspects. For example, both APIs provide documentation, but GHTorrent documentation seems to be out of date. Finally, we conclude the master work by evaluating the queries technologies in a real case study. We collect the top-10K GitHub repositories based on the number of stars to answer three research questions about open source software development in different countries. In this study, we analyze the results achieved by using data from different APIs. We identify some variation in the results, although most of them can be related to the time difference in data collection of both APIs. As practical contributions of this work, we present analyzes of each of the APIs to assist researchers when choosing an API for data collection.

Palavras-chave: GitHub, GitHub REST API, GHTorrent, Open Source development.

List of Figures

3.0	Documentation for <i>/search/repositories</i> endpoint.	52
3.1	Textual description of <i>projects</i> table from GHTorrent dump.	54
3.2	Example query from GHTorrent documentation.	54
4.1	Popularity in terms of the number of stars using data from GitHub REST API	72
4.2	Popularity in terms of the number of stars using data from GHTorrent . .	72

List of Tables

2.1	Analysis of GitHub APIs.	23
3.1	Analysis of GitHub REST API, GitHub Archive, and GHTorrent usage on the selected papers.	26
3.2	Data collected from selected paper.	27
3.3	Queries implemented using GitHub REST API and GHTorrent. If the query was successfully implemented we mark with ✓; if not, we mark with ✗. . .	55
3.4	Number of endpoints and requests for each GitHub REST API query, and number of tables and joins used to implement each query for GHTorrent. .	57
3.5	Finding from GitHub REST API and GHTorrent.	62
4.1	Top-20 countries with more repositories using GitHub REST API and GHTorrent.	68
4.2	Top-3 language by country using GitHub REST API and GHTorrent. . . .	71

Contents

1	Introduction	14
1.1	Motivation	14
1.2	Proposed Work	15
1.3	Contributions	16
1.4	Publications	17
1.5	Outline of the Dissertation	18
2	Background	19
2.1	GitHub REST API	19
2.2	GitHub GraphQL	20
2.3	GHTorrent	21
2.4	GitHub Archive	22
2.5	Final Remarks	23
3	Comparison	25
3.1	Study Design	25
3.2	Queries	28
3.3	Discussion and Lessons Learned	49
3.3.1	Initial Setup	49
3.3.2	Documentation	51
3.3.3	Data Availability	54
3.3.4	Queries Complexity	56
3.3.5	Limitations	58
3.4	Limitations	60
3.5	Final Remarks	61
4	Case Study	63
4.1	Research Questions	63

4.2	Study Design	64
4.2.1	GitHub REST API	64
4.2.2	GHTorrent	64
4.3	Queries	65
4.3.1	GitHub REST API	65
4.3.2	GHTorrent	66
4.3.3	Conclusion	67
4.4	Results	67
4.5	Limitations	73
4.6	Final Remarks	73
5	Conclusion	75
5.1	Overview	75
5.2	Contributions	75
5.3	Related Work	77
5.4	Future Work	78
	Bibliography	79

Chapter 1

Introduction

1.1 Motivation

Nowadays, GitHub is the most popular platform for source code storage and sharing, with more than 100 million repositories and 40 million users.¹ GitHub is not only used to store and share code but also to organize the contributions from developers around the world. We can find popular Open Source Software (OSS) in GitHub including Linux², OpenOffice³, and Notepad++.⁴ These facts along with the friend API provided by GitHub contribute to the platform being commonly used in Software Engineering research [Cosentino et al., 2017]. Due to the large interest in GitHub for research, other platforms that collect and share GitHub data were created, such as GitHub Archive⁵ and GHTorrent [Gousios and Spinellis, 2012]. Essentially, these systems collect data from GitHub and make it available for further analysis.

Each one of the APIs provides GitHub data in different formats. GitHub REST API⁶ is the official GitHub API that provides access to GitHub through endpoints that returns the results in JSON format. This API provides data about repositories, users, issues, pull requests, among others. Recently, GitHub announced a new API version that uses a data query language called GraphQL.⁷ This version allows the user to specify the data to be returned, also in a JSON format. GitHub Archive⁵ is a project created by Ilya Grigorik that collects GitHub events and makes them available

¹<https://github.com/features>

²<https://github.com/torvalds/linux>

³<https://github.com/apache/openoffice>

⁴<https://github.com/notepad-plus-plus/notepad-plus-plus>

⁵<https://www.gharchive.org/>

⁶<https://developer.github.com/v3/>

⁷<https://developer.github.com/v4/>

in hourly archives. Users can easily retrieve this data via an HTTP client and use it in their work. Although GitHub Archive does not provide direct data about repositories and users, they can be obtained through the collected events. The last project is GHTorrent [Gousios and Spinellis, 2012] created by Georgios Gousios. This project provides an off-line mirror of GitHub data, which is populated with data from GitHub events and GitHub API. The data is available in structured and unstructured formats. In the first case, the data is organized in tables on MySQL; in the latest one, the raw data is stored in MongoDB.

When collecting GitHub data, it is important to comprehend these different APIs, so that we can adopt the most suitable option. However, **we still lack studies about these APIs, especially studies analyzing their differences and target audience.**

1.2 Proposed Work

This master dissertation aims to provide a comparative analysis of two popular APIs for accessing GitHub data: GitHub REST API and GHTorrent. We are not comparing GitHub Archive because, based on the papers we analyzed to extract the queries, we did not find studies mentioning GitHub Archive as the API used for data collection. Also, this study does not include GitHub GraphQL API because it is a recent API not yet widely adopted.

We first conduct a study with queries that are implemented for both APIs to compare them under different dimensions. Finally, we analyze the APIs through a case study about open source software development in different countries. Both of these works are described next.

Query implementation and analysis. We implement the example queries for both APIs to exemplify and explain their usage. These queries were extracted from papers of two major Software Engineering conferences: International Conference on Software Engineering (ICSE, 2017 edition) and Mining Software Repositories Conference (MSR, 2017 edition). Based on the implemented queries, we compare both APIs under different dimensions:

1. **Examples.** To analyze the APIs, we propose 23 real-world queries that we attempt to implement in this study for GitHub REST API and GHTorrent. We not only implement each query but also explain how they work and expose the implementation decisions.

2. **Initial Setup.** An API might require an initial configuration effort before being ready for use. Based on our experience while implementing the queries, we explain the process of the initial setup for both APIs, concluding with a comparison between them.
3. **Documentation.** While implementing the queries, it is important to have access to their documentation. We provide insights about the documentation of each API, regarding its availability, organization, completeness, update status, and examples.
4. **Data.** When we start a research, it is important to know which data is accessible via each API. Based on the implemented queries, we provide insights related to the GitHub data provided by each one.
5. **Query Complexity.** Users need to be able to easily build a query to the API. Thus, based on different aspects of query implementation, we analyze and compare the complexity of the queries manually implemented in this master dissertation.
6. **Limitations.** It is important to know the limitations of an API before starting using it. Therefore, based on our experience while working with them, we present the limitation we faced when using both APIs.

Case Study. We also compare GitHub REST API and GHTorrent implementing a study about open source software development in different countries. We compare both APIs under different aspects:

1. **Study Design.** We present the study design specific for the different APIs.
2. **Queries.** To analyze the case study, we implement and explain the queries for both APIs. We conclude with a discussion about the differences between them.
3. **Results.** We analyze and compare the results of the proposed research questions for each API.

1.3 Contributions

We highlight the following contributions of the studies presented in this master dissertation:

- We implement 23 real queries using GitHub REST API and 17 real queries using GHTorrent extracted from two relevant Software Engineering conferences: ICSE, 2017 and MSR, 2017.
- We provide several insights about different dimensions for each API: initial setup, documentation, data availability, query complexity, and limitations. We also provide a comparative discussion of each dimension.
- We provided a detailed case study about Open Source Software development around the world replicated using GHTorrent and GitHub REST API.
- We provide a discussion about the differences in the results of a study performed using GitHub REST API and GHTorrent.

1.4 Publications

This master dissertation produced the following publications, and, therefore, it contains material of them:

- Mombach, T., and Valente, M. T. (2018). GitHub REST API vs GHTorrent vs GitHub Archive: A Comparative Study. In *6th Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM)*, p. 1-8.

The following publications represent other research efforts during this masters work:

- Ferreira, M., Mombach, T., Ferreira, K., and Valente, M. T. (2019). Algorithms for Estimating Truck Factors: A Comparative Study. In *Software Quality Journal*, vol. 1, pages 1-37.
- Brito, G., Mombach, T., and Valente, M. T. (2019). Migrating to GraphQL: A Practical Assessment. In *26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 140-150, 2019.
- Mombach, T., Ferreira, M., Valente, M. T., and Ferreira, K. (2017). Caracterização do Papel Desempenhado por Desenvolvedores Responsáveis pelo Truck Factor de Projetos de Software. In *5th Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM)*, p. 1-8.

1.5 Outline of the Dissertation

The remaining of this dissertation is organized as follows:

- **Chapter 2** provides an overview of APIs that provide access to GitHub data. First, we present the purpose of GitHub REST API, GitHub GraphQL API, GHTorrent, and GitHub Archive, and how they work.
- **Chapter 3** presents the queries for GitHub REST API and GHTorrent derived from papers of two major Software Engineering conferences. First, we explain how we selected these queries. Then, we present the implementation for each one of the 23 queries. Finally, we describe our findings and lessons learned based on the process of implementing the queries.
- **Chapter 4** describes a case study about open source software development around the world. We replicate this study for GitHub REST API and GHTorrent. We present the study design, queries, and results for both APIs. We finish this chapter comparing both APIs regarding query complexity and result equivalence.
- **Chapter 5** presents the contributions of this dissertation, including related work, limitations, and future work.

Chapter 2

Background

In this chapter, we provide an overview of GitHub REST API, GitHub GraphQL API, GHTorrent, and GitHub Archive, and conclude with a comparative analysis of them. In the master dissertation, we will refer to these system as APIs, even though GHTorrent and GitHub Archive are datasets.

2.1 GitHub REST API

In this section, we discuss about the GitHub REST API since it is a consolidated API used in a large amount of research studies in Software Engineering.

GitHub REST API (v3) is based on a commonly used architectural style for implementing web applications, called *Representational State Transfer* (REST). As a result, GitHub provides a set of endpoints to access different types of data available through HTTP requests to <https://api.github.com/>. For example, we can perform a request to <https://api.github.com/repos/:owner/:repo> to retrieve data about *:owner/:repo* repository. A large variety of data are accessible through GitHub REST API through different endpoints, for example: user, team, project, pull requests, issues and so on.

The data returned by GitHub REST API endpoint is in JSON format and it might contain multiple results that can be divided into pages, which can be retrieved using *?page* and *per_page* parameters. By default, each page contains 30 items, but for some endpoints this number can be increased up to 100 using *?per_page* parameter. For example, the request to https://api.github.com/repos/:owner/:repo/issues?&page=2&per_page=100 retrieves 100 issues from the second page of results for *:owner/:repo* repository. When using pagination, each response contains a link header with access links for immediate next page (*next*), immediate previous page (*prev*), first

page (*first*) or last page of results (*last*). Besides pagination, it is also possible to specify parameters for some GitHub REST API requests, for example `https://api.github.com/repos/:owner/:repo/issues?sort=comments&direction=asc` contains parameters to specify that the results will be sorted by number of comments (`sort=comments`) and in ascendant order (`direction=asc`).

Although a large amount of public data is available through REST API endpoints, there is a limitation regarding the amount of data that can be accessed per hour. This limit is calculated based on the number of requests and it varies depending on the authentication provided by the user: user and password, token, or key/secret. Currently, the rate limit for authenticated and unauthenticated requests is 5,000 and 60 requests per hour, respectively. However, this limit is different for the search endpoints, where it is 30 requests per minute for authenticated requests and 10 for unauthenticated.

2.2 GitHub GraphQL

In September 2016, GitHub announced GitHub API v4 that uses GraphQL query language proposed by Facebook.⁸ Unlike REST API, this version provides only one endpoint where the query in the request body defines the operation to be executed. GitHub GraphQL API is accessible through an HTTP request to `https://api.github.com/graphql`. This endpoint allows requesting data using *query* or update data in the server using *mutation*.

The GraphQL API has a schema to represent the data available through objects. The schema specification is accessible in the documentation and through queries to the endpoint since the API is introspective. As the name suggests, GitHub GraphQL API has a structure similar to a graph. The objects that represent entities from GitHub are the *nodes* of the graph, which can be connected by edges when an object has *connections* with other objects. For example, the object *Repository* represents a GitHub repository, and a repository can have stars, forks, issues and pull requests that are represented by other objects and accessible through connections. This schema organization allows the user to navigate through the objects and their connections to retrieve all necessary data in a single request.

For this API, version we have two types of limits: node limit and rate limit. The rating limit is 5,000 points per hour, which is calculated based on the number of nodes in the query result. For the node limit, a connection can return up to 100 nodes, and the complete query cannot exceed 500,000 nodes. To overcome this last limit, we can

⁸<https://github.blog/2016-09-14-the-github-graphql-api/>

use pagination to traverse through all the necessary data. For pagination, we have *first* or *last* arguments to retrieve up to 100 results along with *offset* or *after* to traverse through the results.

2.3 GHTorrent

GHTorrent is a project created and maintained by Georgios Gousios [Gousios and Spinellis, 2012]. It provides an off-line mirror of GitHub data based on the events collected from the events endpoint and complementing it with further requests to GitHub REST API. GHTorrent provides access to all collected GitHub events, as well as a structured version generated from parsing these events.

GitHub Events API provides more than 40 different event types for activities in the website, but it is only possible to access the last 300 public events that occurred in the last 90 days. Moreover, GitHub REST API has a limitation of 5,000 requests per hour for authenticated requests, as mentioned in Section 2.1. These are limitations faced by the ones that rely on GitHub REST API, and to overcome them, GHTorrent uses donated tokens to perform more than 5,000 requests per hour. It also relies on cache to avoid duplicate requests, and it uses distributed data collection to ensure that no event will be lost.

GHTorrent provides the collected data in two different formats: MySQL and MongoDB dumps. MySQL dumps are made available monthly and contain the structured data parsed from GitHub events organized into tables that represent entities such as projects, users, organizations, commits, pull requests, issues, and watchers. In this format, GHTorrent also provides geolocation information (city, state, and country) about GitHub users based on free-text location available on their profiles.

MongoDB dumps are made available daily with the raw data collected from GitHub events and their dependencies. These dumps provide a GitHub event history, which is an information that is not provided by the official GitHub API. However, differently from MySQL dumps, the provided data is the actual JSON returned from the events endpoint, not a refined version.

The data provided by GHTorrent can be accessed in three different ways: MySQL or MongoDB dumps, web service provided by GHTorrent, or Google Big Query when accessing from MySQL. The first option is best suited for users that need to perform a major data analysis. The second and last options are best suited for users that need to process a small amount of data, since, when using GHTorrent web service, the computational resources are shared among all users. Moreover, when using Google Big

Query, it is only possible to process 1TB of data free of charge.

GHTorrent's effort of collecting GitHub events started on February 2012, and even though it helps researchers to overcome some GitHub REST API limitations, such as API rate limit and restricted number of events, it also has some limitations itself. GitHub REST API does not support deletion events and since GHTorrent relies on events to generate the structured data, it may not reflect the actual GitHub state. For example, there is an event for starring a repository, but there is no correspondent event for unstar, so the number of stars from a repository will never decrease. Another problem is that the data is collected using GitHub API, and this API may change along the time, causing problems to GHTorrent which needs to adapt its code to the new API. For example, the events were provided by the Timeline API in the past, and then changed to Events API. Another problem is that GitHub does not provide access to the whole event history, so any problem during event collection might cause losing the event forever.

2.4 GitHub Archive

GitHub Archive is a project created by Ilya Gregorik to collect and store public events from GitHub and make them available to the community. These events are collected using the events endpoint, and the collected JSONs are made available through files that can be downloaded from GitHub Archive⁹ website.

GitHub Archive started collecting GitHub events in December 02, 2011, using the currently deprecated Timeline API. In January 01, 2015, the events started to be collected using the Events API, which provides access to more than 20 different events from GitHub. This data is collected in JSON format, and it is provided to the users in hourly archives that can be downloaded using an HTTP client.

In addition to the hourly archives, the collected events are also publicly available on Google Big Query platform, where the data is stored in tables organized by year, month, and day, and it is also updated hourly. These tables have columns with basic information about the repository, the author of the event, and a JSON object for the payload that can be manipulated using native JSON functions from Google Big Query.

GitHub Archive, as well as GHTorrent, provides access to GitHub events history that is not provided by the official GitHub API. However, GitHub Archive provides the whole data collected only in hourly files, so it is necessary to download one by one. This process is simplified by Google Big Query, where the data is better organized

⁹<https://www.gharchive.org/>

(since data is stored in tables instead of files) and there is no need to download it; although, it is only possible to process 1 TB of data for free.

2.5 Final Remarks

	GitHub REST API V3	GitHub GraphQL API V4	GitHub Archive	GHTorrent
Data Since	-	-	02/12/2011	2012
Data From	-	-	GitHub REST API Events	GitHub REST API
Data Type	GitHub Data	GitHub Data	GitHub Events	Structured GitHub Data
Data Update	Live	Live	Hourly	Monthly

Table 2.1: Analysis of GitHub APIs.

The first aspect that must be taken into account is the date range we want to collect data. When using the official GitHub REST API or GitHub GraphQL API, the information available covers all GitHub history, but this does not happen when using GitHub Archive and GHTorrent. GitHub Archive collects events since 02/12/2011 and GHTorrent since 2012. Therefore, even though it has a large amount of data, it does not contain the whole history of events from GitHub, since GitHub was officially launched in April 2008.¹⁰

Another important aspect is who maintains the projects. GitHub REST API and GraphQL API are maintained by GitHub. GHTorrent was created by Georgios Gousios as an Open Source Project, and it is maintained by him and an open source community. The same is true for GitHub Archive created by Ilya Grigorik. Since GitHub Archive and GHTorrent are not maintained by GitHub, they have to access GitHub data through GitHub APIs; indeed, both projects collect data using REST API. GitHub Archive access the Events API directly, while GHTorrent uses other APIs besides the Events API. To overcome the rate limit problem, GHTorrent asks for token donations from the community.

In other words, GitHub Archive and GHTorrent store part of the data available on GitHub and to make it easier for researchers to access this data without a rate limit. GitHub Archive stores data from events that occurs on GitHub organized in hourly

¹⁰<https://github.com/about/milestones>

files accordingly to the time they happened. GHTorrent provides structured data as MySQL dumps, which are monthly released.

Chapter 3

Comparison

3.1 Study Design

GitHub has become a fundamental source of information for Software Engineering research due to the large amount of open source projects it hosts and to the easy data access through its API. To properly compare the APIs usage, we decided to rely on actual GitHub data collection queries from papers published on two relevant Software Engineering conferences: International Conference on Software Engineering (ICSE, 2017 edition) and Mining Software Repositories Conference (MSR, 2017 edition), with 68 and 37 accepted papers, respectively.

Therefore, we analyzed a total of 105 papers using the following procedure:

1. First, we searched each paper for the keywords *GitHub*, *GitHub REST API*, *GHTorrent*, and *GitHub Archive*.
2. When these keywords were found, we checked whether GitHub data is used in the study by reading the papers' abstract and analyzing the context where the keyword was found.

After this inspection, 12 papers were selected for our study, where 3 of them are from ICSE and 9 from MSR. For each paper, we identified the API used to collect data. In cases where the API was not mentioned, we considered that they were using GitHub REST API, since it is the official tool provided by GitHub to collect data and it was already identified as the most popular tool for that purpose [Cosentino et al., 2016]. Table 3.1 contains the number of papers by conference that rely on GitHub REST API, GHTorrent and GitHub Archive to collect data. All ICSE papers use GitHub REST

API to collect data, 5 MSR papers use GitHub REST API and 4 use GHTorrent, but none of the selected papers mentioned the use of GitHub Archive for data collection.

Conference	Total	GitHub REST API	GHTorrent	GitHub Archive
ICSE	3	3	0	0
MSR	9	5	4	0

Table 3.1: Analysis of GitHub REST API, GitHub Archive, and GHTorrent usage on the selected papers.

Once the papers were selected, we carefully read them to identify the data collected. Most papers describe in the *Methodology* or *Study Design* sections how GitHub data was collected. For example, a paper [Gharehyazie et al., 2017a] selected for the study a list of non-fork GitHub projects implemented in Java that have at least 2 contributors, more than 10 commits and that are at least 1 year old, as reported in its *Methodology* section:

"We selected **Java projects that had at least 2 developers, were at least 1 year old, and had more than 10 commits**. These criteria remove smaller and younger projects, most of which have a single developer, and usually do not contribute much to the ecosystem, and would have skewed our results [7]. **We also eliminated projects that were forked using GitHub interfaces**, as they would highly skew our findings due to their duplicate code."

In Table 3.2, we show the excerpt about the data collected for each paper in this study. For papers using GHTorrent, we can see that in most the cases is only being used to identify repositories based on characteristics. Similarly, GitHub REST API is used to collect repositories based on characteristics, but it is also used to collect issues and pull requests. Based on this excerpts, we identified the data collected for each paper and proposed queries to collect the required data. In addition, the queries we extracted might be shared by more than one of the selected papers. For example, Query #18 retrieves top- x repositories implemented in language y , and it is shared by two papers that reported to collect the top 100 C repositories [Floyd et al., 2017] and top five most starred Java repositories on GitHub [Xiong et al., 2017]. We use these queries to explain how to use GitHub REST API and GHTorrent through data collection using these APIs, and compare them based on their characteristics and limitations. These queries are not implemented for GitHub Archive because we did not find any paper that report using it. Also, we did not include GitHub GraphQL API because it is a recent API not widely adopted yet.

Paper	Tool	Data Collected	Query	
Decoding the representation of code in the brain: an fMRI study of code review and expertise (ICSE 2017)	GitHub REST API	<i>"A pool of candidate pull requests were selected by considering the top 100 C repositories on GitHub as of March 2016 and obtaining the 1,000 most recent pull requests from each. We considered the pull requests in a random order and filtered to consider only those with at most two edited files and at most 10 modified lines, as well as those with non-empty developer comments"</i>	Query #18	
			Query #12	
			Query #13	
Precise condition synthesis for program repair (ICSE 2017)	GitHub REST API	<i>"The first dataset consists of the top five most starred Java projects on GitHub as of Jul 15th, 2016"</i>	Query #18	
How do Developers Fix Cross-project Correlated Bugs? A case study on the GitHub scientific Python ecosystem (ICSE 2017)	GitHub REST API	<i>"The column #devs presents the number of members in the organizations. [...] The next four columns show the total numbers of commits, branches, releases, and contributors at the time of March 12th, 2016."</i>	Query #11	
			Query #4	
			Query #9	
			Query #8	
			Query #7	
		<i>First, for any of the studied projects, we collected all its closed bugs.</i>	Query #14	
Exception Evolution in Long-lived Java Systems (MSR 2017)	GitHub REST API	<i>"Using the GitHub search facility, we query for repositories that are (i) written in Java, (ii) created before 1 January 2012, (iii) are starred more than 10 times and have been forked at least 10 times, (iv) are greater than 1MB in size, and (v) have at least one commit since 1 July 2016."</i>	Query #21	
How Open Source Projects use Static Code Analysis Tools in Continuous Integration Pipelines (MSR 2017)	GitHub REST API	<i>"we ranked the selected projects in terms of popularity by using the GitHub APIs"</i>	Query #3	
Extracting Build Changes with BUILDDIFF (MSR 2017)	GitHub REST API	<i>"We retrieved a list of Java projects ordered by their star rating and removed projects that do not use MAVEN as build system and projects with less than 3500 commits in the repository or rated with less than 1000 stars"</i>	Query #22	
Bug Characteristics in Blockchain Systems: A Large-Scale Empirical Study (MSR 2017)	GitHub REST API	<i>"We focus on closed bug reports from the issue repositories, i.e., closed issues with tag bug. We exclude open bug reports because they are not fixed and may not have enough information for our analysis, or they may not even be bugs. TABLE I shows the number of closed bug reports for the eight blockchain projects"</i>	Query #14	
			Query #15	
			Query #3	
			Query #7	
			Query #8	
			Query #1	
Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub (MSR 2017)	GHTorrent	<i>"we selected all projects that are not forks themselves, and received more than 50 stars."</i>	Query #19	
			<i>"For each project, we extracted five GITHUB features from GHTORRENT: main project language $\in C, C++$, Java, Ruby, ..., number of watchers $\in [51; 41,663]$, number of external contributors $\in [0; 2,986]$, number of pull requests $\in [0; 27,750]$, number of issues $\in [0; 127,930]$ and active years of the project $\in [0; 45]$;"</i>	Query #1
			Query #3	
			Query #7	
			Query #5	
			Query #6	
			Query #2	
Some From Here, Some From There: Cross-Project Code Reuse in GitHub (MSR 2017)	GHTorrent	<i>"We selected Java projects that had at least 2 developers, were at least 1 year old, and had more than 10 commits. [...] We also eliminated projects that were forked using GitHub interfaces, as they would highly skew our findings due to their duplicate code."</i>	Query #23	
Stack Overflow in Github: Any Snippets There? (MSR 2017)	GHTorrent	<i>"... we downloaded 909k Python non-fork repositories based on the GHTorrents metadata..."</i>	Query #17	
Choosing an NLP Library for Analyzing Software Documentation: A Systematic Literature Review and a Series of Experiments (MSR 2017)	GitHub REST API	<i>"we also wrote a script to facilitate the random sampling of README files from Java related GitHub projects"</i>	Query #10	
			Query #16	
Structure and Evolution of Package Dependency Networks (MSR 2017)	GHTorrent	<i>"For Rust, we cloned all projects listed in GHTorrent, but for JavaScript and Ruby, we only cloned those that either had at least one fork or at least one star..."</i>	Query #17	
			Query #20	

Table 3.2: Data collected from selected paper.

In Section 3.2, we present and explain GitHub REST API and GHTorrent queries to collect the desired data based on the discussion around Table 3.2. When going through this section, it is important to have in mind that a query might be shared by more than one paper. Consequently, the result of these queries can differ from the results from the paper we used to extract the query.

In Section 3.3, based on our effort while constructing and explaining the queries from Section 3.2. We present our findings about the usage of each API regarding environment setup, documentation, data availability, query construction and limitations, and compare each API based on these dimensions.

3.2 Queries

Query #1: Main programming language of a repository

A repository on GitHub can use multiple languages, but in this query we retrieve the programming language mostly used to implement repository *example/repo*.

GitHub REST API

For this query, we also use `/repos/:owner/:repo` endpoint since one of the properties in the JSON result is the main programming language used by the repository (*language*).

```
https://api.github.com/repos/example/repo
```

GHTorrent

When using GHTorrent, this data can be directly retrieved from the *language* column in *projects* table.

```
1 SELECT language FROM projects
2 WHERE url = 'https://api.github.com/repos/example/repo'
```

Another possibility is to retrieve the most used programming language from *project_languages* table. This table stores all languages used by the repository along with the number of bytes implemented in that language. It is possible to exist multiple entries of the same language for a project, but the timestamp of when the entry was added (*created_at*) can differentiate them. We did not use this approach because requires access in two tables.

Query #2: Age of a repository

In this query, we calculate the age of a repository based on the creation date retrieved.

GitHub REST API

We continue to use `/repos/:owner/:repo` endpoint without any parameters besides the repository name in this query. The JSON result includes the creation date of a repository (`created_at`). Since this endpoint does not return the age directly, it requires an additional step to calculate the age based on the creation date.

```
https://api.github.com/repos/example/repo
```

GHTorrent

For GHTorrent, we need to query the `projects` table, which contains the repository creation date in the `created_at` column. We use specific functions to calculate the age based on the date in the `created_at` column, not requiring an additional step after the data is collected.

```
1 SELECT FLOOR(DATEDIFF(CURRENT_TIMESTAMP(),created_at)/365) AS age
2 FROM projects WHERE url = 'https://api.github.com/repos/example/repo'
```

Query #3: Number of stars of a repository

In this query, we calculate the number of stars for `example/repo` repository.

GitHub REST API

The `repos/:owner/:repo` endpoint, which is used in the following query, fetches repository data, including the number of stars in `stargazers_count` property. For this query we use no parameters.

```
https://api.github.com/repos/example/repo
```

GHTorrent

To calculate the number of stars using GHTorrent, we need two tables: `projects` and `watchers`. We use `projects` table to retrieve the repository identifier (lines 2-3), which is

used to fetch from *watchers* table the ids of the users that gave a star to the repository. As a result, we count the number of distinct user identifiers to get the number of stars (line 1). This is necessary because a user can appear twice in the list since they can star and unstar a repository several times and there is not event for unstar.

```

1 SELECT COUNT(S.user_id) FROM ( SELECT user_id FROM watchers w
2 WHERE repo_id = (SELECT id FROM projects WHERE
3 url = 'https://api.github.com/repos/example/repo')
4 GROUP BY user_id ) S

```

Query #4: Number of contributors of a repository

A repository contributor in GitHub is a user that did at least one commit to the repository. Therefore, in this query, we need to retrieve the number of contributors of *example/repo*.

GitHub REST API

GitHub REST API provides *repo/:owner/:repo/contributors* endpoint that lists all contributors of a repository. Therefore, we use this endpoint along with *page* and *per_page* to calculate the number of contributors.

When we use pagination, the response header of a request might contain a *Link* property with queries to the first, next and last page of results. In this example, we request the first page of results with only one contributor. If the repository have more than one contributor, we retrieve, from the *Link* property in the response header, the query to the last page of results to determine the total number of contributors. The *page* property of the query to the last page of results is sufficient to retrieve the number of contributors because each page contains only one result.

```
https://api.github.com/repos/example/repo/contributors?&page=1&per_page=1
```

GHTorrent

In this query for GHTorrent, we use *projects*, *commits*, and *project_commits* tables. The *project_commits* table is used along *projects* table to identify the commits of a repository (lines 4-6), since forked repositories can have shared commits. These two tables along with *commits* table is used to identify the authors of commits (*author_id*)

in a repository. We count the number of distinct authors to retrieve the number of contributors of a repository (line 1).

```

1 SELECT  COUNT(X.author_id) AS num_contributors FROM (
2 SELECT  c.author_id FROM commits c
3 INNER JOIN
4 (SELECT commit_id FROM project_commits
5 WHERE project_id = (SELECT id FROM projects
6 WHERE url = 'https://api.github.com/repos/example/repo')) pc
7 ON c.id = pc.commit_id
8 GROUP BY c.author_id ) x

```

Query #5: Number of commits in a repository

We retrieve the number of commits in repository *example/repo* with this query.

GitHub REST API

The endpoint *repo/:owner/:repo/commits* returns a list of commits in a repository. Therefore, we use this endpoint along with *page* and *per_page* parameters to request the first page containing one result. If the repository has more than one commit, we use the query to the last page of results in the response header to calculate the number of commits following the same approach from Query #4.

```
https://api.github.com/repos/example/repo/commits?&page=1&per_page=1
```

GHTorrent

In this query, we use *projects* and *project_commits* tables to construct a GHTorrent query. The *projects* table is used to retrieve the repository identifier (lines 3-4), which is used to count the number of commits in a repository using *project_commits* table (line 1).

```

1 SELECT COUNT(commit_id) AS num_commits
2 FROM project_commits
3 WHERE project_id = (SELECT id FROM projects WHERE
4                   url = 'https://api.github.com/repos/example/repo')

```


Query #6: Number of pull requests of a repository

In this query, we retrieve the number of pull requests of *example/repo* repository.

GitHub REST API

We use the *repo/:owner/:repo/pulls* endpoint with *page* and *per_page* parameters to request the first page with one result for this query. If more than one result is found, the response header has the *Link* property with the query to the last page of results. We use this query to calculate the number of pull requests; as explained in Query #4.

```
https://api.github.com/repos/example/repo/pulls?&page=1&per_page=1
```

GHTorrent

For GHTorrent, we use *pull_requests* table to count the number of pull requests (line 1) referring to the repository identified using the *projects* table (lines 2-3).

```
1 SELECT COUNT(pullreq_id) AS num_pull_requests FROM pull_requests
2 WHERE base_repo_id = (SELECT id FROM projects WHERE url =
3 'https://api.github.com/repos/example/repo')
```

Query #7: Number of issues from a repository

For this query, we retrieve the number of issues in *example/repo*.

GitHub REST API

For this query, we use two endpoints. First, we use *repo/:owner/:repo/issues* endpoint with *per_page* and *page* parameters besides the repository name. The number of issues is calculated based on the query to the last page in the *Link* property in the response header; as discussed for Query #4.

```
https://api.github.com/repos/example/repo/issues?&page=1&per_page=1
```

Every pull request is an issue on GitHub, so the result of the first query returns the number of issues plus pull requests. To retrieve only the number of issues, besides the first query, we need to execute Query #6 to retrieve the number of pull requests.

Therefore, the difference between the result of these two queries is the number of issues in a pull request.

GHTorrent

In this query, we use the *issue* table to identify and count the issues (line 1) from the repository with GHTorrent identifier retrieved from *projects* table (lines 2-3). In addition, since every pull request is an issue for GitHub, we filter out the pull requests (line 4)

```
1 SELECT COUNT(issue_id) AS num_issues FROM issues
2 WHERE repo_id = (SELECT id FROM projects WHERE
3 url = 'https://api.github.com/repos/example/repo')
4 AND pull_request_id IS NULL
```

Query #8: Number of releases of a repository

In this query, we retrieve the number of releases of *example/repo* repository.

GitHub REST API

The *repo/:owner/:repo/releases* endpoint provides access to a list of published releases of a repository. We use this endpoint along with *page* and *per_page* parameters to calculate the number of releases based on the *Link* property in the response header; as described in Query #4.

```
https://api.github.com/repos/example/repo/releases?&page=1&per_page=1
```

GHTorrent

Although GHTorrent provides access to a large amount of data from GitHub, there is no data related to repository releases. As a result, we are not able to implement this query for GHTorrent.

Query #9: Number of branches of a repository

In this query, we collect the number of branches in *example/repo* repository. A branch is a parallel version of a repository that does not affect the primary branch.

GitHub REST API

The `repo/:owner/:repo/branches` endpoint retrieves a list of branches from a repository, and in this query is used along with `page` and `per_page` parameters. The number of branches is calculated based on the `Link` property in the response header; as discussed for Query #4.

```
https://api.github.com/repos/example/repo/branches?&page=1&per_page=1
```

GHTorrent

GHTorrent does not store data about branches, so it is not possible to implement a query to retrieve the desired data.

Query #10: README of a repository

Each repository on GitHub can have a README file that is used to explain the purpose of the project and how to use it. In this query, we want to retrieve this file for `example/repo` repository.

GitHub REST API

The `repo/:owner/:repo/readme` endpoint from GitHub REST API retrieves data about the repository README. From the JSON result of this request, we need the data in the `content` property. The content data is encoded, so we have an additional step of decoding it.

```
https://api.github.com/repos/example/repo/readme
```

GHTorrent

GHTorrent does not provide data about README files, so we cannot implement a GHTorrent query to retrieve the data.

Query #11: Number of members in the organization

Besides user account, GitHub has organization accounts, which are shared accounts managed by GitHub users identified as members. In this query, we retrieve the number of members in an organization called `example`.

GitHub REST API

To calculate the number of members in an organization the `orgs/:org/members` endpoint is used along with `per_page` and `page` parameters. The result is the first page containing only one result. If there is more than one member in the organization, the number of members is extracted from the query to the last page of results available in the `Link` property from the response header; as explained for Query #4.

```
https://api.github.com/orgs/example/members?&per_page=1&page=1
```

GHTorrent

GHTorrent stores not only information about users, but also organizations. In this query, we use the tables `users` and `organization_members` to calculate the number of users that are members of the organization. Particularly, we count the distinct users (line 1) in `organization_members` table for the organization identified using `users` table (line 3).

```
1 SELECT COUNT(DISTINCT user_id) as num_members
2 FROM organization_members
3 WHERE org_id = (SELECT id FROM users WHERE login = 'example')
```

Query #12: X most recent pull requests from a repository

In this query, we retrieve the GitHub identifier of the 1,000 most recent pull requests for a repository `example/repo`.

GitHub REST API

The endpoint `repos/:owner/:repo/pulls` provides access to a list of pull requests of a repository based on the specified parameters. In this query, we use `state` parameter to retrieve pull requests in all states, also `sort` and `direction` parameters to retrieve the pull requests in a descendant order of the creation date. The `per_page` and `page` parameters are used to iterate through 10 pages, each one with 100 results. We collect only the `full_name` information from each result.

```
https://api.github.com/repos/example/repo/pulls?&state=all&sort=created &direction=desc&page=1&per_page=100
```

GHTorrent

GHTorrent stores the data collected about pull requests in four tables. We use two of them, *pull_requests* and *pull_request_history*, along with the *projects* table to implement this query. We use *projects* table to retrieve the GHTorrent identifier (*id*) of the repository, which is used to identify its pull requests (*id* and *pullreq_id*) stored in the *pull_requests* table (lines 2-5). Then, the result of this subquery is combined with the 1,000 most recent pull requests resulted from subquery of *pull_request_history* table that order the results based on the creation date (*created_at*) of the *opened* event (*action*) of a pull request (lines 7-10). The final results is ordered and limited to 1,000 pull requests (lines 9-10).

```

1 SELECT pr.pullreq_id
2 FROM (SELECT id, pullreq_id FROM pull_requests
3       WHERE base_repo_id =
4             (SELECT id FROM projects
5              WHERE url = 'https://api.github.com/repos/example/repo')) pr
6 INNER JOIN
7 (SELECT pull_request_id, created_at FROM pull_request_history
8  WHERE action = 'opened'
9  ORDER BY created_at DESC
10 LIMIT 1000 ) prh
11 ON pr.id = prh.pull_request_id
12 ORDER BY prh.created_at DESC

```

The repository name is the concatenation of the owner's login and repository name. As a result, to identify a repository, we can use the *users* and *projects* tables, that contains the owner login (*login*) and project name (*name*), respectively. However, on the *projects* table, we have a *url* column that stores the URL used by GHTorrent to collect repository information, and in the URL we have the complete repository name. As a result, instead of accessing two tables, we use this *url* column from *projects* (line 5).

Query #13: Number of edited files, modified lines and comments in a pull requests

In this query, we need to collect the number of edited files, modified lines (additions and deletions) and comments from pull request #753 of *example/repo* repository.

GitHub REST API

The endpoint `repos/:owner/:repo/pulls/:pr_number` returns data about a given pull request. For this query, we do not use any parameter. The result contains a set of data related to the pull request, but we only need the data from properties: `changed_files`, `additions`, `deletions`, and `comments`.

```
https://api.github.com/repos/example/repo/pulls/753
```

GHTorrent

GHTorrent provides data about pull requests, but this data is not sufficient to implement a query that retrieves the required data from pull requests. GHTorrent does not provide data related to the number of edited files or modified lines on a pull request.

Query #14: Issues on state *y* and label *x* of a repository

In this query, we retrieve the GitHub identifier of all closed issues with label `bug` from repository `example/repo`.

GitHub REST API

GitHub REST API provides `/repos/:owner/:repo/issues` endpoint to fetch issues data of a repository according to a set of parameters. For this query, we use `state` and `labels` parameters to filter closed issues with a label `bug`. From the JSON result, we need to retrieve only the value from property `id`.

```
https://api.github.com/repos/example/repo/issues?&state=closed&labels=bug
&page=1&per_page=100
```

In GitHub, all pull requests are issues, so this endpoint might return both pull requests and issues. To differentiate them, we also need to check for a `pull_request` key, which appears only on pull request results.

GHTorrent

In this query, we use five tables distributed in sub-queries that are combined to retrieve the desired result. We filter out pull request (`pull_request_id`) from the result using `issues` table (line 14). The `projects` table is used to retrieve the GHTorrent identifier for the repository (lines 2-4). After that, tables `repo_labels` and `issue_labels` are used

to identify issues labeled as *bug* (lines 5-8). In the last step, table *issue_events* is used twice to identify issues with no reopened event after the close event (lines 9-13). The result of those joins generates a list of issues from *issues* table that meet the specified criteria.

```

1 SELECT issue_id FROM issues i
2 INNER JOIN project p
3 ON i.repo_id = p.id
4 AND p.url = 'https://api.github.com/repos/example/repo'
5 INNER JOIN issue_labels il
6 ON il.issue_id = i.id
7 INNER JOIN repo_labels rl
8 ON il.labels_id = rl.id AND rl.name = 'bug'
9 INNER JOIN issue_event ie1
10 ON i.id = ie1.issue_id AND ie1.action = 'closed'
11 LEFT JOIN issue_event ie2
12 ON ie1.issue_id = ie2.issue_id AND ie2.action = 'reopened'
13 AND ie2.created_at > ie1.created_at
14 WHERE i.pull_request_id IS NULL AND ie2.issue_id IS NULL

```

Query #15: Title, body and comments of an issue

In this query, we collect the title, body and comments message from issue #123 of *example/repo*. For the comments, the necessary information is the body of the comment.

GitHub REST API

The `/repos/:owner/:repo/issues/:issue_number` endpoint returns data about an issue, such as *title* and *body*. In this example, we use this endpoint with no parameters.

```
https://api.github.com/repos/example/repo/issues/123
```

After collecting the title and body of an issue from *title* and *body* properties in the JSON result of the first query, we use `/repos/:owner/:repo/issues/:issue_number/comments` endpoint to retrieve the comments of a given issue. We add *page* and *per_page* parameters to request the first page of results with 100 comments. For issues with more than 100 comments, the response header has a *Link* property in the header that contains the next page of results.

Therefore, we should use this property to iterate through the complete list of comments. From each result, we retrieve the data in the *body* property.

```
https://api.github.com/repos/example/repo/issues/123/comments?&page=1
&per_page=100
```

GHTorrent

Although GHTorrent stores data about issues, it is not possible to implement a query to return all necessary data. GHTorrent does not store the title, body, or comments messages in the existing databases.

Query #16: Repositories mainly implemented in programming language *x*

In this first query, we retrieve a list of repositories names that have Java as their main language. The repository name is a combination of the owner's name along with the repository name.

GitHub REST API

For this query, we use the *search/repositories* endpoint that provides an optimized search service for repositories based on pre-defined criteria. In this example, the query uses a *language* qualifier to retrieve only repositories whose main programming language is Java. The *fork* qualifier is added because only non-forked repositories are returned by default in this endpoint, and there is no such criteria for this query. These two qualifiers are specified in the required parameter *q* that is used to specify search keywords and qualifiers.

```
https://api.github.com/search/repositories?q=language:java+fork:true&page=1
&per_page=100
```

This search query might have multiple results, but due to a limitation in the search endpoints, it is only possible to return 1,000 for the same search query. The results are organized into pages, and we use *per_page* parameter to define up to 100 results on each page and *page* parameter to navigate through them. The result for this endpoint has a set of data related to the repositories that fulfill the search criteria, but for this query we only need the complete name stored in the property *full_name*.

GHTorrent

On GHTorrent, the *projects* table stores data about GitHub repositories. In this example, we use two columns from *projects* table: the *language* column, which stores the most used programming language of the repository, and the *deleted* column, which identifies whether the repository still exists on GitHub. This table is used to identify Java repositories that still exist on GitHub.

The result of this query is a list of URLs (line 1) of GitHub repositories that fulfill the specified criteria (line 2). The URL has the format *https://api.github.com/repos/:owner_login/:repo_name*. As a result, it is possible to extract the repository name from it, but it requires an additional step. Also, it is possible to use *projects* and *users* tables to retrieve the repository name, but we use the *url* column to avoid accessing an additional table. This approach is used in all the following queries when it is necessary to retrieve the repository name.

```
1 SELECT url FROM projects
2 WHERE UPPER(language) = 'JAVA' AND deleted = 0
```

Query #17: Non-fork repositories mainly implemented in programming language x

In this query, we retrieve the name of Python repositories that are not forked from another.

GitHub REST API

In this example, we use *search/repositories* only with the *language* qualifier since this endpoint removes forked repositories from the results by default. Also, we have *page* and *per_page* parameters to navigate through the results that contain a set of data about a repository. Similarly to the other queries, we need only the data from *full_name* property. Due to the search endpoints limitation, it is possible to retrieve only 1,000 results for this query.

```
https://api.github.com/search/repositories?q=language:python&page=1
&per_page=100
```

GHTorrent

The *projects* table is used in this query since it stores not only the main language used by a repository (*language*), but also the id of the parent for forked repositories (*forked_from*). For active repositories that fulfill the criteria (lines 1-2), we return the URL (*url*) that can be used to acquire the full repository name (line 1).

```
1 SELECT url FROM projects WHERE deleted = 0
2 AND forked_from IS NULL AND UPPER(language) = 'PYTHON'
```

Query #18: Top-x repositories mainly implemented in programming language y

The goal for this query is to retrieve the top-100 Java repositories according to the number of stars. Regarding the results, the only required information from each repository is its complete name.

GitHub API

This query also uses the *search/repositories* endpoint to search for repositories that fulfill the query specification. The *language* qualifier is an element of the *q* parameter to filter repositories mainly implemented in Java. We use *sort* and *order* parameters to sort the result in a descendant order according to the number of stars. Since we need to retrieve the first 100 results of this search query, we use the *page* and *per_page* parameters to request the first page with 100 results. For each result, we only need the data in the *full_name* property.

```
https://api.github.com/search/repositories?q=language:java+fork:true
&sort=stars&order=desc&page=1&per_page=100
```

GHTorrent

In the GHTorrent query, *projects* and *watchers* tables are combined to generate the desired result. The *projects* table holds repository data such as repository language (*language*) and state (*deleted*) that are used to identify active Java GitHub repositories (lines 2-3). The number of stars is retrieved based on data from *watchers* that stores when a user (*user_id*) give a star to a repository (*repo_id*) (lines 5-6). The result for

this query is the URL of the first 100 selected repositories (line 9) according to the number of stars (line 8). The URL is used to retrieve the repository name.

The *watchers* table is populated with data from events generated when a user gives a star to a repository on GitHub. A user can also remove a star from a repository, but there is no event for this action. As a result, it is not possible to remove an invalid star from *watchers* table which keeps invalid data. For this query, we consider each user once while calculating the number of stars for a repository, since a user can give and remove a star many times.

```

1 SELECT p.url
2 FROM (SELECT id, url FROM projects WHERE deleted = 0
3       AND UPPER(language) = 'JAVA') p
4     INNER JOIN
5     (SELECT repo_id, COUNT(DISTINCT user_id) AS num_watchers
6       FROM watchers GROUP BY repo_id) w
7     ON p.id = w.repo_id
8 ORDER BY w.num_watchers DESC
9 LIMIT 100

```

Query #19: Non-fork repositories with more than x stars

This query retrieves the name of repositories with more than 50 stars that are not fork from another one.

GitHub REST API

The *search/repositories* is used only with the *stars* qualifier to select repositories with more than 50 stars since this endpoint returns only non-fork repositories by default. In addition to the qualifier parameter (*q*), *page* and *per_page* parameters were added to iterate through the results. It is only possible to retrieve 1,000 results for this query, and the only data necessary from each result is stored in the *full_name* property.

```
https://api.github.com/search/repositories?q=stars:>50&page=1 &per_page=100
```

GHTorrent

In the GHTorrent query, the *projects* table has information about the status of the repository (*deleted*) and whether it is a fork (*forked_from*), whereas the *watchers* table

contains the relation between the user and the starred repositories. We exclude from the results projects that were deleted from GitHub or that are forks in the subquery with *projects* table (lines 2-3). Moreover, we calculate the number of stars by counting the number of unique users for each repository in the *watchers* table (lines 5-6). This is done because a GitHub user can give and remove a star several times and GitHub REST API does not provide events for removing a star from a repository. The result of these two subqueries are combined and repositories with 50 stars or less are excluded (line 8).

```

1 SELECT p.url
2 FROM (SELECT id, url FROM projects
3       WHERE deleted = 0 AND forked_from IS NULL) p
4     INNER JOIN
5     (SELECT repo_id, COUNT(DISTINCT user_id) AS num_watchers
6       FROM watchers GROUP BY repo_id) w
7     ON p.id = w.repo_id
8 WHERE w.num_watchers > 50

```

Query #20: Get non-fork repositories implemented in programming language x with at least y forks and z stars

The goal is to retrieve the name of non-fork repositories implemented in JavaScript with at least one fork and one star.

GitHub REST API

The *search/repositories* endpoint allows filtering repositories based on characteristics using qualifiers. We use *language*, *stars*, and *forks* qualifiers to retrieve JavaScript repositories with at least one fork and one star, and forked repositories are excluded from the result by default. Furthermore, we also use the *per_page* and *page* parameters to traverse over the 10 pages of results, which returns 100 repositories each. The only data necessary from each result is the *full_name*.

```

https://api.github.com/search/repositories?q=language:javascript+
forks:>=1+stars:>=1&page=1&per_page=100

```

GHTorrent

For the GHTorrent query, we use *projects* table in two subqueries. The first subquery retrieves non-fork repositories (*forked_from*) mainly developed in JavaScript (*language*) that exists in GitHub (*deleted*) (lines 2-3). The second is used to calculate the number of forked repositories (*id*) originated from another repository (*forked_from*) excluding deleted forks (*deleted*) (lines 9-10). Also, the *watchers* table is used in a third subquery to calculate the number of stars in a repository by counting the number of unique users (*user_id*) that starred a repository (*repo_id*) (lines 5-6). The results of the three subqueries are combined removing repositories with less than one fork or one star from the final result (line 12).

```

1 SELECT p1.url
2 FROM (SELECT id, url FROM projects WHERE forked_from IS NULL and
3       deleted = 0 AND UPPER(language) = 'JAVASCRIPT') p1
4     INNER JOIN
5       (SELECT repo_id COUNT(DISTINCT user_id) AS num_watchers
6        FROM watchers GROUP BY repo_id) w
7     ON p1.id = w.repo_id
8     INNER JOIN
9       (SELECT forked_from, COUNT(id) AS num_forks FROM projects
10      WHERE deleted = 0 GROUP BY forked_from) p2
11    ON w.repo_id = p2.forked_from
12 WHERE p2.num_forks >= 1 AND w.num_watchers >= 1

```

Query #21: Non-fork repositories implemented in a specific language with at least x contributors, more than y commits and at least z year old

In this query, we retrieve the name of non-fork repositories implemented in Java by at least two contributors, with more than 10 commits and at least one year old.

GitHub REST API

We continue to use the *search/repositories*, but together with two other endpoints: *repo/:owner/:repo_name/commits* and *repo/:owner/:repo_name/contributors*. For *search/repositories*, the *language* and *created* qualifiers are used to filter repositories

based on the main language and creation date, since forked repositories are filtered out by default. In addition, we use *per_page* and *page* parameters to iterate through the results.

```
https://api.github.com/search/repositories?q=language:java+created:<2018-11-04
&page=1&per_page=100
```

For each repository retrieved by the previous query, we calculate the number of commits in the repository using Query #5. Finally, to calculate the number of contributors for each repository we use Query #4.

GHTorrent

The *projects* table holds information about GitHub repositories as discussed for previous queries, and in this query we use the main language (*language*), fork (*forked_from*), creation date (*created_at*), and the is deleted (*deleted*) information to filter repositories (lines 1-3). The *project_commits* table stores the relationship between repository (*project_id*) and commits (*commit_id*) even for forked repositories, and the *commits* table has the information about the author of a commit (*author_id*). We join these two tables to calculate the number of commits and the number of contributors in a repository (lines 5-9). After these subqueries are constructed, we combine the results and filter out repositories that do not have at least 2 contributors and more than 10 commits (line 11).

```
1 SELECT p.url FROM (SELECT id, url FROM projects WHERE deleted = 0
2     AND forked_from IS NULL AND UPPER(language) = 'JAVA'
3     AND created_at < DATE_SUB(NOW(), INTERVAL 1 YEAR)) p
4 INNER JOIN
5     (SELECT pc.project_id, COUNT(c.id) AS num_commits,
6     COUNT(DISTINCT c.author_id) AS num_contributors
7     FROM (commits AS c1 INNER JOIN project_commits AS pc
8     ON c1.id = pc.commit_id)
9     GROUP BY pc.project_id) c
10 ON pc.project_id = c.project_id
11 WHERE pc.num_commits > 10 AND c.num_contributors > 2
```

Query #22: Repositories with more than x commits and y stars implemented in programming language z sorted by the number of stars

We use this query to retrieve the complete name of repositories mainly implemented in Java, with more than 3,500 commits and 1,000 stars, and sorted by the number of stars in a descendant order.

GitHub REST API

The *search/repositories* endpoint provides a list of qualifiers that can be used to filter repositories based on characteristics such as *language* and *stars*. However, there is no qualifier to filter repositories by the number of commits.

For this reason, we use multiple queries to achieve our goal. The first query retrieves repositories mainly implemented in Java that have more than 1,000 stars using *search/repositories* endpoint. The *language* and *stars* qualifiers are used to filter repositories, and the *fork* qualifier is used to include forked repositories in the results. In addition, the *per_page* and *page* parameters are used to iterate through the result pages, along with *sort* and *order* to sort the result by number of stars in a descendant order.

```
https://api.github.com/search/repositories?q=stars:>1000+language:java+fork:true
&sort=stars&order=desc&page=1&per_page=100
```

Then, for each repository retrieved by the previous query, we use Query #5 to calculate the number of commits. If the number of commits is less or equal to 3,500, this repository is removed from the final result.

GHTorrent

In GHTorrent, the table *projects* stores data about the main programming language for GitHub repositories, but not the number of stars or commits, which can be retrieved using tables *watchers* and *project_commits*. In this query, we use *projects* table in a subquery to filter existing repositories based on its main language (lines 2-3). The number of stars is calculated in a subquery using the *watchers* table to count the number of unique users (*user_id*) who starred a repository (*repo_id*) (lines 5-6). The subquery using *project_commits* table is used to calculate the number of commits (*commit_id*) of a repository (*repo_id*) (lines 9-10). The result of these subqueries are combined keeping only repositories with more than 1,000 stars and 3,500 commits (line

12), and sorted in descending order according to the number of stars (lines 13). The final result is the URL of the repositories, which can be used to retrieve its name.

```

1 SELECT p.url
2 FROM (SELECT id, url FROM projects
3       WHERE deleted = 0 AND UPPER(language)='JAVA') p
4     INNER JOIN
5     (SELECT repo_id, COUNT(DISTINCT user_id) AS num_watchers
6       FROM watchers GROUP BY repo_id) w
7     ON p.id = w.repo_id
8     INNER JOIN
9     (SELECT project_id, COUNT(commit_id) AS num_commits
10      FROM project_commits GROUP BY project_id) pc
11    ON w.repo_id = pc.project_id
12 WHERE w.num_watchers > 1000 AND pc.num_commits > 3500
13 ORDER BY w.num_watchers DESC

```

GHTorrent also provides a *commits* table that stores a link to a repository (*project_id*), and one might be confused about why we are not using this table for this query instead of *project_commits*. The *commits* table has a link between the commit and the first project that GHTorrent associated with that commit, so in case of forked projects, this table might not provide the actual number of commits. However, in *project_commits* table, this problem does not occur since it stores the relationship between repositories and commits, including forked projects.

Query #23: Repositories implemented in programming language *u*, created before date *v*, with more than *w* stars, at least *x* forks, size greater than *y* and at least one commit after date *z*

The result for this query is the name of repositories created before January 01, 2012 implemented in Java, with more than 10 stars, at least 10 forks, size greater than 1MB, and at least one commit since July 01, 2016.

GitHub REST API

For this query, we use the *search/repositories* endpoint with qualifiers *language*, *stars*, *forks*, *created*, *size*, and *pushed* to filter projects by most used programming language,

number of stars, number of forks, creation date, size and last commit update. In addition, we use *page* and *per_page* parameters to traverse the result pages with. For each one of the 1,000 results, we need only the data from *full_name* property.

```
https://api.github.com/search/repositories?q=language:java+created:<2012-01-01+pushed:>=2016-07-01+forks:>=10+stars:>10+size:>1000&page=1&per_page=100
```

GHTorrent

```
1 SELECT p1.url FROM ((SELECT id, url, forked_from, language
2 FROM projects WHERE deleted = 0 AND UPPER(language) = 'JAVA'
3 AND created_at < '2012-01-01') p1
4 INNER JOIN
5 (SELECT repo_id, COUNT(DISTINCT user_id) AS num_watchers
6 FROM watchers GROUP BY repo_id) w
7 ON p1.id = w.repo_id
8 INNER JOIN
9 (SELECT forked_from, COUNT(id) AS num_forks FROM projects
10 GROUP BY forked_from) p2
11 ON w.repo_id = p2.forked_from
12 INNER JOIN
13 (SELECT project_id, SUM(bytes) FROM project_languages p11
14 WHERE created_at = (SELECT MAX(created_at) FROM project_languages p12
15 WHERE p11.project_id = p12.project_id) GROUP BY project_id) s
16 ON p2.forked_from = s.project_id
17 INNER JOIN
18 (SELECT pc.project_id, MAX(c.created_at) AS last_commit
19 FROM (commits AS c INNER JOIN project_commits AS pc
20      ON c.id = pc.commit_id)) c1
21 ON p2.forked_from = c1.project_id
22 WHERE p2.num_forks >= 10 AND w.num_watchers > 10
23 AND c1.last_commit >= '2016-07-01' AND s.size > 1000000
```

For GHTorrent query we used five different tables to retrieve all the necessary data to filter the repositories based on the specified criteria. We used the table *projects* to retrieve repositories based on language and creation date (lines 1-3) and to calculate

the number of forks (lines 9-10). The table *watchers* was used to calculate the number of stars of repositories (lines 5-7), and table *project_languages* was used to calculate the size of repositories (lines 13-15). In addition, tables *commits* and *project_commits* tables are used together to retrieve the date of last commit (lines 18-20). Finally, the result of all this subqueries are merged and we filter the repositories based on the number of forks, number of watchers, date of last commit and size to obtain the repositories that meet the specified criteria.

3.3 Discussion and Lessons Learned

Previously in this chapter, we presented example queries for GitHub REST API and GHTorrent. We based these examples on real data collection scenarios from two relevant Software Engineering conferences. In this section, we describe our findings and lessons learned about such APIs, based on the queries implemented in Section 3.2. For each API, we analyze the following aspects: initial setup, query complexity, data processing, documentation, and limitations. We end the analysis of each aspect with a comparative evaluation of GitHub REST API and GHTorrent.

3.3.1 Initial Setup

In this section we discuss the setup process for GitHub REST API and GHTorrent based on the effort we put to setup the APIs to construct and test the implemented queries.

3.3.1.1 GitHub REST API

GitHub REST API provides access to its data through endpoints that can be easily accessed using HTTP requests. As a result, the GitHub REST API requires no specific configuration in the environment for data collection. However, it might be necessary to use authentication to be able to request data depending on the number and type of requests to the API. For GitHub REST API, we can authenticate via username and password or OAuth tokens. For the implemented queries, we used OAuth tokens in the request header to authenticate the requests.

Each GitHub user can generate multiple authentication tokens with access permissions based on the selected scopes.¹¹ To generate these tokens in the GitHub website, we can go to the personal access tokens section under developer settings, and,

¹¹<https://developer.github.com/apps/building-oauth-apps/understanding-scopes-for-oauth-apps/#available-scopes>

there we can define its identifier and scopes. After that, GitHub does not allow access to the token again, but it is possible to update the scopes for the token or delete it accessing the token identifier. In case of loss, it is necessary to generate a new token that invalidates the old one. For our queries, we need to access only the public data available on GitHub, so the token does not require any scope.

Even though we do not need any special permission to access the data required for the implemented queries, the number of requests increases from 30 to 5,000 requests per hour when using authentication. However, the tokens of the same user share this limit, as it is per user, not per token.

3.3.1.2 GHTorrent

GHTorrent provides access to its data in three different manners: Web service provided by GHTorrent, Google Big Query, and dumps. We analyze the initial setup for MySQL dumps since it is the one used in this study.

When using MySQL dumps, a large amount of data needs to be downloaded, stored, and processed. These dumps are a large compressed collection of CSV files. For example, the size of the last dump available on the GHTorrent website (June 2019) is 102,973 MB. After downloading the dump, we need to extract everything from the compressed file before storing it in the database. This task might require significant data processing and data storage.

After downloading and extracting the data, we have to populate the database. Each CSV file extracted from the dump is a table in the database, and it can have more than 100 GB. As a result, we need a machine with considerable storage space and processing capacity to populate the database. Moreover, GHTorrent provides a script to create all the necessary entities and to insert the data into the database. It also provides instructions to restore individual tables. Depending on the target machine, the complete database setup can take days or even weeks to finish. In our study, after weeks of trying to insert the data, the database was ready to use.

3.3.1.3 Conclusion

When comparing the initial setup based on our experience, GitHub REST API requires almost none, unlikely GHTorrent, which requires a great deal of effort. For GHTorrent, it is necessary to download, store, and process a large amount of data before starting using the tool. In this regard, it is much more complex to use GHTorrent since the environment setup requires much more time and space when compared to the GitHub REST API.

3.3.2 Documentation

During the process of understanding the tools and implementing queries, it is very important to have easy access to precise and complete documentation. In this section, we analyze these aspects regarding the documentation provided for GitHub REST API and GHTorrent based on our perception while using them to implement the queries presented in Section 3.2.

Search repositories 

Note: The `topics` property for repositories on GitHub is currently available for developers to preview. To view the `topics` property in calls that return repository results, you must provide a custom `media` type in the `Accept` header:

```
application/vnd.github.mercy-preview+json
```

Warning: The API may change without advance notice during the preview period. Preview features are not supported for production use. If you experience any issues, contact [GitHub Support](#).

Find repositories via various criteria. This method returns up to 100 results [per page](#).

When searching for repositories, you can get text match metadata for the `name` and `description` fields when you pass the `text-match` media type. For more details about how to receive highlighted search results, see [Text match metadata](#).

```
GET /search/repositories
```

(a) Note and warning.

Parameters

Name	Type	Description
<code>q</code>	string	Required. The query contains one or more search keywords and qualifiers. Qualifiers allow you to limit your search to specific areas of GitHub. The REST API supports the same qualifiers as GitHub.com. To learn more about the format of the query, see Constructing a search query . See "Searching for repositories" for a detailed list of qualifiers.
<code>sort</code>	string	Sorts the results of your query by number of <code>stars</code> , <code>forks</code> , or <code>help-wanted-issues</code> or how recently the items were <code>updated</code> . Default: <code>best match</code>
<code>order</code>	string	Determines whether the first search result returned is the highest number of matches (<code>desc</code>) or lowest number of matches (<code>asc</code>). This parameter is ignored unless you provide <code>sort</code> . Default: <code>desc</code>

(b) Parameters.

3.3.2.1 GitHub REST API

GitHub provides an official documentation page for GitHub REST API that can be easily accessed and contains information about all endpoints present in this API.¹²

The documentation is divided into sections that in most cases are groups of endpoints based on similar characteristics. For example, the search section lists all endpoints that can be used to search for GitHub entities. For each endpoint, we have a detailed explanation, in most cases including information about parameters and an example. For example, for the endpoint to search for repositories (`/search/repositories`¹³), which was widely used in Section 3.2, is a good example of this. It is first provided a brief explanation about the purpose of the endpoint, along with notes and limitations (Figure 3.1a). Then, we can find a detailed explanation about `q`, `sort`, and `order` parameters (Figure 3.1b). The last part is an example query for this endpoint along with its response (Figure 3.0c).

Although GitHub provides precise and complete documentation about the endpoints in the REST API, the large variety of endpoints to collect the same type

¹²<https://developer.github.com/v3/>

¹³<https://developer.github.com/v3/search/#search-repositories>

Example

Suppose you want to search for popular Tetris repositories written in Assembly. Your query might look like this.

```
curl https://api.github.com/search/repositories?q=tetris+language:assembly&sort=stars&or
```

You can search for multiple topics by adding more `topic:` instances, and including the `mercypreview` header. For example:

```
curl -H "Accept: application/vnd.github.mercy-preview+json" \
https://api.github.com/search/repositories?q=topic:ruby+topic:rails
```

In this request, we're searching for repositories with the word `tetris` in the name, the description, or the README. We're limiting the results to only find repositories where the primary language is Assembly. We're sorting by stars in descending order, so that the most popular repositories appear first in the search results.

```
Status: 200 OK
Link: <https://api.github.com/resource?page=2>; rel="next",
      <https://api.github.com/resource?page=5>; rel="last"
X-RateLimit-Limit: 20
X-RateLimit-Remaining: 19

{
  "total_count": 40,
  "incomplete_results": false,
  "items": [
    {
      "id": 3081286,
      "node_id": "MDEwOJlJlcG9zaXRvcnkzMDgxMjg2",
      "name": "Tetris",
      "full_name": "dtrupenn/Tetris",
      "owner": {
        "login": "dtrupenn",
        "id": 872147,
        "node_id": "MDQ6VXNlcjg3MjE0Nw==",
        "avatar_url": "https://secure.gravatar.com/avatar/e7956084e75f239de85d3a31bc172",
        "gravatar_id": "",
        "url": "https://api.github.com/users/dtrupenn",
        "received_events_url": "https://api.github.com/users/dtrupenn/received_events",
        "type": "User"
      },
      "private": false,
      "html_url": "https://github.com/dtrupenn/Tetris",
      "description": "A C implementation of Tetris using Pennsim through LC4",
      "fork": false,
      "url": "https://api.github.com/repos/dtrupenn/Tetris",
      "created_at": "2012-01-01T00:31:50Z",
      "updated_at": "2013-01-05T17:58:47Z",
      "pushed_at": "2012-01-01T00:37:02Z",
      "homepage": "",
      "size": 524,
      "stargazers_count": 1,
      "watchers_count": 1,
      "language": "Assembly",
      "forks_count": 0,
      "open_issues_count": 0,
      "master_branch": "master",
      "default_branch": "master",
      "score": 10.309712
    }
  ]
}
```

(c) Example of result.

Figure 3.0: Documentation for `/search/repositories` endpoint.

of data can be a problem in some cases. For example, for Query #1 we can retrieve the main programming language of a repository using `/repos/:owner/:repo` or `/repos/:owner/:repo/languages` endpoints. The same applies for Query #7 that lists the issues of a repository, where it is possible to use `/search/issues` or `/repos/:owner/:repo/issues`. These endpoints might return the same data, but they have different parameters and limitations. Therefore, to select the endpoint that fits best to the use case, it is necessary to check the parameters allowed for the endpoint, its limitations, and the complete data in the response. In this case, the documentation does not help, because sometimes the endpoints returning similar data might not even be in the same section. For example, `/search/issues` and `/repos/:owner/:repo/issues` endpoints, the first one is in the search section and the second in the issues section. This is not an easy task since a slight change in your use case might impact the endpoint used to collect data.

GitHub REST API also provides an overview section presenting basic aspects of the API and its use. For example, there is documentation regarding authentication, rate limit, and pagination, which is very helpful when starting to use the API.

3.3.2.2 GHTorrent

In this section, we analyze the documentation related to the MySQL database dumps provided by GHTorrent. Since GHTorrent requires an environment setup before starting using it, we first analyze the documentation about this initial step. In the downloads page of GHTorrent, we can find that they provide a script to restore the database and instructions on how to do use it. Also, the script and documentation are available in the dump file. It is possible to restore the complete database using the script or restore individual tables using the provided commands.

Besides the *downloads* tab, the GHTorrent web page also includes a *docs* tab that provides access to documentation for MongoDB and MySQL dumps. For MySQL, we have access to the architectural diagram containing the database schema with all the tables, columns and their relationships.¹⁴ In addition to the database schema, GHTorrent also provides a brief text description of the data stored on each table and column. For example, the table *projects* has a brief explanation about the data that it stores and about three out of its ten columns, as can be observed in Figure 3.1. It is clear that not every column requires explanation, for instance, an *id* column. However, an explanation for column *language* would be valuable for us while implementing Query #1 since a repository can be implemented in more than one language.

¹⁴<http://ghtorrent.org/files/schema.pdf>

projects

Information about repositories. A repository is always owned by a user.

- The `forked_from` field is empty unless the project is a fork in which case it contains the `id` of the project the project is forked from.
- The `deleted` field means that the project has been deleted from Github.
- The `updated_at` field indicates when the last full update was done for this project.

Figure 3.1: Textual description of *projects* table from GHTorrent dump.

Also, there is a difference in the database schema and the textual documentation. The *repo_milestones* table in the database schema does not have a textual description.

GHTorrent also provides some example queries in the documentation. Three along with the textual description of tables in *Entities and their relationships* section, and four other examples in *Example Queries* section. However, these examples contain just a statement about the data to be collected and the MySQL query implementation, as presented in Figure 3.2.

Example queries

List commits for a repository

```
select c.*
from commits c, project_commits pc, projects p, users u
where u.login = 'rails'
      and p.name = 'rails'
      and p.id = pc.project_id
      and c.id = pc.commit_id
order by c.created_at desc
```

Figure 3.2: Example query from GHTorrent documentation.

3.3.2.3 Conclusion

Both GitHub REST API and GHTorrent provide documentation about the data available, including how it is organized, how it can be accessed, and examples. Nonetheless, based on our experience, both have issues that could be improved. Despite that, GitHub REST API is more complete when compared to GHTorrent since GHTorrent has a very brief textual explanation about the tables and a small number of examples.

3.3.3 Data Availability

Based on the proposed queries from Section 3.2, we discuss about the data available for GitHub REST API and GHTorrent.

Query	GitHub REST API	GHTorrent
Query #1	✓	✓
Query #2	✓	✓
Query #3	✓	✓
Query #4	✓	✓
Query #5	✓	✓
Query #6	✓	✓
Query #7	✓	✓
Query #8	✓	✗
Query #9	✓	✗
Query #10	✓	✗
Query #11	✓	✓
Query #12	✓	✓
Query #13	✓	✗
Query #14	✓	✓
Query #15	✓	✗
Query #16	✓	✓
Query #17	✓	✓
Query #18	✓	✓
Query #19	✓	✓
Query #20	✓	✓
Query #21	✓	✓
Query #22	✓	✓
Query #23	✓	✓

Table 3.3: Queries implemented using GitHub REST API and GHTorrent. If the query was successfully implemented we mark with ✓; if not, we mark with ✗.

3.3.3.1 Github REST API

GitHub REST API is the official API provided by GitHub to access public GitHub data through endpoints. As presented in Table 3.3, we were able to implement all the proposed queries using these endpoints. As a result, based on the data collection for this study, there is no limitation regarding data availability for GitHub REST API.

3.3.3.2 GHTorrent

GHTorrent consumes data from GitHub REST API to create the monthly MySQL dumps. However, GHTorrent does not retrieve and store all the data available through the GitHub REST API. As a result, not all GitHub REST API queries can be implemented using GHTorrent. For GHTorrent, we were not able to implement 5 out of the 23 proposed queries (21.7%), as listed in Table 3.3. Based on these five queries, we identify some of the data that is not available in GHTorrent. Regarding Query #13, although GHTorrent stores data about pull requests, there is no data related to the

number of edited files and modified lines for a pull request. From these two queries, we conclude that GHTorrent does not include data about files stored in GitHub. In Query #15, we need to recover the title, body, and comments of an issue, but *issues* and *issue_comments* tables do not store these type of data. For Queries #8 and #9, we could not retrieve the number of branches and releases, respectively, because not all GitHub entities are present in GHTorrent dumps. In Query #10, we need access to the README file present in GitHub repositories, but there is no file from GitHub stored since this is not the purpose of GHTorrent.

3.3.3.3 Conclusion

GitHub REST API, which is the official API to collect data from GitHub, provides access to every GitHub public data we need to access for queries in Section 3.2. Moreover, GHTorrent provides a partial offline mirror of GitHub public data since we could not implement all proposed queries for GHTorrent. Based on the queries that we could not implement for GHTorrent, we can conclude that it does not store data about files and some GitHub entities. We can conclude that GHTorrent stores information about the relationship between important GitHub objects, but not actual data.

Finally, it is import to highlight that GHTorrent provides access to historical data through its monthly dumps. Using GitHub REST API we cannot access the state of GitHub in previous months as we can do with GHTorrent dumps.

3.3.4 Queries Complexity

In this section, we evaluate the complexity of implementing the queries from Section 3.2. For GitHub REST API, we analyze the number of endpoints and requests, and, for GHTorrent, the number of tables and joins in the query. We use these metrics of complexity because the more endpoints and tables we have to use, the more data we need to understand and process to get to the desired result.

3.3.4.1 GitHub REST API

GitHub REST API queries have a low complexity regarding the number of endpoints as 19 of them (82%) use only one endpoint, as presented in Table 3.4. When analyzing the number of requests, it varies according to the number of results since it is only possible to request up to 100 results per page. According to Table 3.4 most queries (52%) execute only one request since for a considerable number of requests we just need to retrieve a single result or use the data in the response header. For example,

Query	GitHub REST API		GHTorrent	
	#endpoints	#requests	#tables	#joins
Query #1	1	1	1	0
Query #2	1	1	1	0
Query #3	1	1	2	0
Query #4	1	1	3	1
Query #5	1	1	2	0
Query #6	1	1	2	0
Query #7	2	2	2	0
Query #8	1	1	N/A	N/A
Query #9	1	1	N/A	N/A
Query #10	1	1	N/A	N/A
Query #11	1	1	2	0
Query #12	1	*	3	1
Query #13	1	1	N/A	N/A
Query #14	1	*	5	5
Query #15	2	*	N/A	N/A
Query #16	1	10	1	0
Query #17	1	10	1	0
Query #18	1	1	2	1
Query #19	1	10	2	1
Query #20	1	10	2	2
Query #21	3	*	3	2
Query #22	2	*	3	2
Query #23	1	10	5	5

Table 3.4: Number of endpoints and requests for each GitHub REST API query, and number of tables and joins used to implement each query for GHTorrent.

Query #13 that retrieves pull request data and Query #4 that calculates the number of contributors of a repository. We also identify a considerable number of queries executing 10 and * requests, where * is a number that depends on the number of results for the query.

In these cases where the result is divided into pages, we need to use pagination and a new request to each new page we need to collect data. This increases the complexity since we have to understand and handle correctly the access to the result pages. The complexity also increases when a query uses more than one endpoint. However, for most queries the number of endpoints and request is one, so the overall complexity is low.

3.3.4.2 GHTorrent

When analyzing the complexity of constructing the queries for GHTorrent, we take into account the number of unique tables and the joins we used to gather the result. For the number of tables, most of the queries use two tables (44%), as a large number of queries in our study retrieve data specific to a repository. Therefore, they first need to retrieve the repository *id* to query other tables that have the data. For example, to calculate the number of stars on Query #3, we first need to access the *projects* table to retrieve the repository *id*, and, then, count the number of stars using table *watchers*. There are queries implemented that are not using two tables, we have four queries using one table, and another four queries using only three tables, and two query using five tables, as presented in Table 3.4. As a result, the complexity to build a query using GHTorrent is acceptable as in half of the cases we need only two tables.

We also analyze the number of *joins*, since it adds extra complexity to the queries. There are nine queries that need joins (50%), and the queries with more than one table are using subqueries. For the queries with joins, 44% of them have just one join, 33% have two joins, and 22% have 5 joins as listed in Table 3.4.

As a result, we conclude that the overall complexity for GHTorrent is low since most queries require up to two tables and no joins.

3.3.4.3 Conclusion

When analyzing the query complexity for GitHub REST API, we used the number of endpoints and requests as metrics. We found that in most cases we could retrieve all the necessary data using only one endpoint and requests. As a result, we conclude that the complexity of these proposed queries is low.

For GHTorrent, we analyzed the queries regarding the number of tables and joins. In most cases, it was used two tables without joins to retrieve the results. From 18 queries, we used joins in nine of them, and the number of joins varies between one and five. Therefore, we can conclude that the overall complexity is low. However, when comparing the number of endpoints and the number of tables, we can see that we complexity of GHTorrent is slightly higher.

3.3.5 Limitations

In this section, we present the limitations faced while implementing the proposed queries.

3.3.5.1 GitHub REST API

GitHub REST API can be used to access a large amount of data. However, we faced some limitations to retrieve the data due to the limit of requests to the API. For *search* endpoints, the limit is 30 requests per minute and for other endpoints, it is 5,000 requests per hour when using authentication.

The *search* endpoints have a limitation which they can only return up to 1,000 results for the same search query, even though the query might contain more results specified in *total_count* field in the response body. For queries that allow sorting the result, it is possible to overcome this limitation by changing the requests to sort and retrieve the results in parts using different search queries. For example, when searching for repositories, specify that the result should be ordered by the number of stars and the range for the number of stars of a repository. Moreover, we can update this range to create a new search query to retrieve another 1,000 results. The downside is that it adds additional complexity because we need to handle not only pagination but also the range value of the search query. Another limitation for search endpoints is that the responses might not be complete, due to timeouts while processing the results. In such cases, GitHub returns what was found before the timeout. Also, the *incomplete_results* property in the result is set to true to indicate to the users that the result might be incomplete.

Another limitation around GitHub REST API, it does not provide access to historical data, i. e., it is not possible to access a previous state of GitHub. For example, it is not possible to retrieve the number of stars of a repository from a year ago, because we do not have access to the date when the star was given, and also to the stars that were removed.

3.3.5.2 GHTorrent

When implementing the queries for GHTorrent, the first limitation we faced was the documentation. Although GHTorrent provides documentation, it is not detailed, complete and up to date, since the last documentation update reported on the website was from 2015.

When discussing the dumps, it is important to remember that they are made available monthly, so they might not be an up to date mirror of GitHub REST API. Moreover, updating the database with new dumps can be very costly.

Since GHTorrent provides monthly dumps and the older ones are still accessible, GHTorrent provides historical data from GitHub by month, different from GitHub REST API, where we can only access the current state of GitHub. However, these

dumps started to be available in 2012, so there is no historical data from previous years of GitHub.

Regarding the data provided, although GHTorrent aims to be an offline mirror of data provided by the Github REST API, as presented in its official web page, not all data available in this API is available on GHTorrent. There are some cases where this is perfectly understandable. For example, there are endpoints in GitHub REST API that we can use to query files that are stored in GitHub, as we discuss in query #10. However, there are examples where it is not possible to find data about specific entities using GHTorrent. For example, about branches and releases for queries #8 and #9.

3.3.5.3 Conclusion

In this section, we presented some specific limitations regarding GitHub REST API and GHTorrent based on our experience while constructing queries. We could see that both APIs have limitations but in different aspects. This information is very important when choosing an API for a study.

3.4 Limitations

The study presented in this section has the following limitations:

- To compare GitHub REST API and GHTorrent, we implemented queries based on a set of papers collected from two Software Engineering conferences in 2017. We identified papers that collected data from GitHub. In some cases, the API used was reported by the paper, but for the papers that did not report the tool, we assumed they were using GitHub REST API. We chose GitHub REST API as default API because studies had pointed out that GitHub REST API was the most used tool. However, this limitation does not have a large impact on the result, because all requests are implemented for both GitHub REST API and GHTorrent.
- The queries presented in Chapter 3 were proposed based on the data collected by each paper selected for this study. As a result, we did not replicate the study from each paper. Instead, we provided queries that could be used to collect the data they reported to be using.
- The queries we presented in Chapter 3 for GitHub REST API and GHTorrent were implemented after long research and experiments using GitHub REST API

and GHTorrent. However, it is possible to have other queries or more efficient ones different from the ones we implemented to retrieve the data for each proposed query.

3.5 Final Remarks

In this chapter, we described the method used to implement actual GitHub data collection queries from papers published on ICSE and MSR conferences. Based on the data collected for the analyzed papers, we proposed 23 queries that were implemented using GitHub REST API and GHTorrent. With these queries, we explained how to use these APIs, how to implement queries, and the particularities of each one of them.

Based on the implementation of these queries for each one of the APIs, we had some interesting insights regarding the initial setup, documentation, data availability, query complexity, and limitations. For the initial setup, it was revealed that GitHub REST API requires almost no initial setup, except for the authentication token. On the other hand, GHTorrent requires a complex initial setup, and very time and space consuming. When analyzing the documentation, we observed that both APIs provide documentation about initial setup, data provided, and examples. However, in our experience, the GitHub REST API is more complete when compared to GHTorrent. Regarding the data availability, we revealed that GHTorrent uses GitHub REST API to collect its data, so every query available for GHTorrent is also available for GitHub REST API. Nevertheless, the opposite is not true because GHTorrent does not collect all data from GitHub REST API. Also, we analyzed the complexity of implementing queries for GitHub REST API and GHTorrent. For GitHub REST API we use the number of endpoints and requests to measure the complexity of queries, and for GHTorrent we used the number of tables and number of joins. The overall complexity of both APIs was acceptable and very similar. In the last section of the results, we presented an analysis of the limitations we faced while using GitHub REST API and GHTorrent. All the main findings are summarized in Table 3.5.

	GitHub REST API	GHTorrent
Initial Setup	Requires no specific configuration in the environment. However, according to the number of requests it might be necessary to create an authentication token.	Complex process since a large dump of the database needs to be downloaded and restored to a MySQL database.
Documentation	Complete documentation organized in sections of related endpoints. For each endpoint, we have a detailed explanation, in most cases including information about parameters and an example.	Architectural diagram of the database and some examples are provided in the documentation. However, the textual explanation of tables and columns is very brief.
Data Availability	All public data from GitHub is accessible through its API.	GHTorrent provides a partial offline mirror of GitHub REST API.
Query Complexity	Low query complexity when analyzing number of endpoints and number of requests. In most cases, the number of endpoints is one and the number of request is at most two.	Low query complexity when analyzing the number of tables and joins. In most cases, the number of tables is at most 2 and joins are only used for 6 queries.
Limitations	Both APIs have limitations regarding different aspects.	

Table 3.5: Finding from GitHub REST API and GHTorrent.

Chapter 4

Case Study

In this chapter, we present a case study using GitHub REST API and GHTorrent to collect data about open source projects from GitHub. We gather data about the top-10K repositories, to provide a study about OSS development around the world. Our goal is to expose and discuss our findings in a study that resembles a real-world application that depends on GitHub data. In this study, we assess not only query complexity and data availability but we also compare the results and study design.

4.1 Research Questions

We retrieve information about the top-10K GitHub repositories according to the number of stars to answer the following research questions about open source software:

1. *How is the distribution of open source software around the world?* We retrieve the number of OSS repositories per country. For GitHub REST API, we were able to identify a country for 5,881 repositories, and for GHTorrent, 6,564 had a country. We identify the top-20 countries based on the number of repositories and use in the remaining research questions.
2. *What are the three most used languages for each country?* We identified the top-3 most popular languages for each country in the top-20. We identified these languages based on the most used language of each repository.
3. *How does the popularity of repositories vary per country?* Since we retrieve the top-10K GitHub repositories, we analyze how the number of stars varies for repositories from countries in the top-20.

4.2 Study Design

We focus on popular repositories selecting the top-10K open-source projects on GitHub by the number of stars. GitHub stars are similar to *likes* in other social networks and they are considered a reliable proxy for the popularity of GitHub repositories [Borges et al., 2016b]. For each repository, we collected the number of stars, the primary language, and the location.

GitHub does not directly provide the geographic location of a project. However, location is a meta-data of GitHub accounts. For example, suppose the project `aserg-ufmg/jscity`. The owner of this project is `aserg-ufmg`, which is an organizational account on GitHub. In `aserg-ufmg`'s profile, it is informed that this organization is located in *Belo Horizonte, Brazil*.¹⁵ Therefore, we consider `aserg-ufmg/jscity` as a Brazilian project. The location is provided as a free text format. In the following sections, we discuss how we collect these data and identify the country for the two APIs studied in this master dissertation.

4.2.1 GitHub REST API

For GitHub REST API, we executed in August 2018 a script implemented to retrieve the data from `search/repositories` to retrieve the top-10K repositories and their information. Based on the location data, this script also attempts to match a project location to a list of country and city names in English; we were successful for 4,216 repositories (42%). Finally, we inspected the location of the remaining repositories, aiming to manually associate them to countries, which was possible in the case of other 1,665 repositories (29%). The remaining 4,119 repositories (41%) include empty location fields, locations which are not countries (e.g., *The Earth*) or locations mentioning more than one country (e.g., *Canada & France*).

After following these steps, we were able to retrieve data from 10,000 repositories and identify the country of 5,881 (59%). This repositories are used to answer the research questions of this study.

4.2.2 GHTorrent

For GHTorrent, we first attempted to use our local MySQL dump from February 2018. However, we tried to retrieve a newer dump for this study. We investigated the possibility of using an online provider of GHTorrent, and we could only access these data in Google Big Query, but the most recent dump is from April 2018. We decided to

¹⁵GitHub repositories have a unique owner, but contributors can be of different countries.

proceed with the dump from April 2018, even though we had a time difference between the data collected using both APIs. The main reason is that the process of restoring the database from a GHTorrent dump is very time and resource consuming.

GHTorrent provides data about users in the *users* table, and it contains a *location* column to store the location provided by the user to GitHub. According to a documentation update from November 2015 about MySQL dumps, five new columns were added to the *users* table to store geographic information: *city*, *state*, *country_code*, *lat*, and *long*. This geographic data is retrieved by GHTorrent using Open Street Maps API, but only for users with non-empty location information. In this study we use data from *country_code* column from *users* table to identify the country of a repository.

Using the data geographic data provided by GHTorrent, we were able to identify the country of 6,564 projects (65%) that are used to answer the research questions.

4.3 Queries

In this section, we present the queries we used for GHTorrent and GitHub REST API to collect the data described in Section 4.2. For this study, we remove forked repositories to avoid repositories with similar characteristics.

4.3.1 GitHub REST API

```
https://api.github.com/search/repositories?q=stars:1..*&sort=stars&order=desc
&page=1&per_page=100
```

To retrieve the top-10K non-fork repositories and the required information about each one of them, we use the *search/repositories* endpoint. As explained in previous sections, this endpoint allows sorting and searching for repositories based on predefined criteria and removes forked repositories from the result by default. Also, a search query can only return up to 1,000 results for the same search query. However, we need 10,000 repositories for this study. As a result, to overcome this limitation, we filter the repositories based on the number of stars and order the result based on the same characteristics, so we can change the range of the number of stars to produce a new query that retrieves another 1,000 repositories. For example, we retrieve repositories with the number of stars from 1 to * in this first query, i.e., all repositories with at least one star. Considering the result is ordered according to the number of stars, we can get the lowest number of stars from the top-1,000 repositories, x , to create a new query that retrieves repositories with 1 to x stars. We follow this process until retrieving all

top-10K repositories, with a cost of not only have to iterate through the result pages but also update the query.

Each result from this request contains data about the number of stars, main language and repository owner of the repository. However, it does not include the owner's location that is retrieved by an additional query.

```
https://api.github.com/users/:owner_login
```

For each repository retrieved by the previous query, we perform a request to the *users* endpoint to collect data about its owner. Among the data returned, we have the *location* property to store the location of the user. We use this data in an additional step to obtain the country of each repository.

Analyzing the number of endpoints, the complexity of this query is acceptable since we use two endpoints to retrieve all the necessary data. However, we also have to consider the number of requests that we need to execute to retrieve the complete result. In the best case scenario, we do at least 100 requests to collect all repositories, but it might require more since we need to handle duplicated repositories due to the change in the query. Also, we execute 10,000 additional requests to retrieve the location of the repository owner. We also have an additional complexity of update not only on the page number but also on the search query to be able to retrieve more than 10,000 repositories using the *search* endpoint.

4.3.2 GHTorrent

To retrieve all the necessary data using GHTorrent, we combine results from different subqueries. The first subquery uses *projects* table to collect the repository names and main language of existing non-fork repositories (lines 3-5). The number of stars is calculated using *watchers* table (lines 7-8). The result of these two subqueries are merged, and we sort this partial result according to the number of stars and retrieve only the first 10,000 repositories (lines 10-11). The top-10K repositories based on the number of stars along with their main language is merged with *users* table to retrieve the user login and country for the owner of each repository in the partial result (lines 13-14) and generate the final result.

```

1 SELECT u.login, pw.name, u.country_code, pw.num_stars, pw.language
2 FROM (SELECT p.name, p.owner_id, p.language, w.num_watchers
3       FROM (SELECT id, name, owner_id, language
4             FROM projects WHERE deleted = 0
5             AND forked_from IS NULL) p
6       INNER JOIN
7         (SELECT repo_id, COUNT(DISTINCT user_id) AS num_watchers
8          FROM watchers GROUP BY repo_id) w
9       ON p.id = w.repo_id
10      ORDER BY w.num_watchers DESC
11      LIMIT 10000) pw
12  INNER JOIN
13    (SELECT id AS user_id, login, country_code
14     FROM users) u
15    ON u.user_id = pw.owner_id

```

Regarding the complexity related to GHTorrent query, we use three different tables to retrieve the complete result, and these tables are combined using two nested joins. To reduce the number of rows that are joined, we first join the necessary column from *projects* with the number of stars calculated for every repository using *watchers* and select only the top-10K repositories. In this way, we can join only the selected repositories with their owners in the *users* table. However, the overall complexity is considerable, due to the number of tables, joins, and filters necessary for this query.

4.3.3 Conclusion

Both APIs have considerable complexity when implementing the queries necessary to retrieve the desired data. GitHub REST API had the additional complexity of not only handle pagination but also updating the search query to overcome the limitation of 1,000 results for search endpoints. GHTorrent provides all the data organized into tables. Therefore, for this query, we used three tables, and we had to handle the joins and filters to calculate the result.

4.4 Results

In this section, we discuss the results for each research question presented in Section 4.1, based on the data collected using GitHub REST API and GHTorrent. Also, we

present a comparative analysis of each research question.

RQ1: How is the distribution of open source software around the world?

GitHub REST API		GHTorrent	
Country	#repositories	Country	#repositories
United States	2,353	United States	2,453
China	1,012	China	884
United Kingdom	350	United Kingdom	325
Germany	294	Germany	270
Canada	202	Canada	182
France	170	France	174
Japan	118	Russia	149
India	92	Japan	117
Netherlands	88	India	99
Australia	86	Australia	90
Sweden	84	Netherlands	88
Spain	81	Sweden	84
Russia	70	Spain	81
Italy	64	Ukraine	63
Ukraine	59	Italy	59
Poland	57	Poland	59
Brazil	52	Austria	51
Switzerland	51	Switzerland	49
Austria	42	Brazil	48
Finland	38	Norway	36

Table 4.1: Top-20 countries with more repositories using GitHub REST API and GHTorrent.

GitHub REST API

To answer this research question, we considered 5,881 projects out of the 10,000 projects (59%) selected because not all of them have valid geographic data. These projects are distributed over 83 countries, as presented in Table 4.1, where the United States has the largest number of projects (2,353 projects, 40%), which is more than two times greater than the second country (China), with 1,012 projects (17%). The difference is even greater between the second and third countries. For the remaining countries in the top-20, the number decreases in an acceptable amount.

GHTorrent

To answer this research question, we considered 6,564 projects out of the 10,000 projects (66%) selected at first because not all of them have valid geographic data. When using GHTorrent, these projects are distributed over 87 countries, as also presented in Table 4.1. The United States has the largest number of projects (2,453 projects, 37%), which is almost three times greater than the second country (China), with 884 projects (13%). From the second to the third country, the difference continues elevated, but it decreases between the remaining countries.

Comparison

When comparing both results, the first aspect to highlight is the number of countries we identified having repositories in the top-10K. Using GHTorrent, we classified repositories to 87 different countries, while using GitHub REST API we identified 83. Using GitHub REST API, we were not able to identify 8 countries identified by GHTorrent. Also, for GHTorrent we could not find repositories for 5 countries present in the result for GitHub REST API.

When comparing the top-20 countries, Table 4.1, we have almost the same countries in both lists, except the last one that is Finland for GitHub REST API and Norway for GHTorrent. For most countries, the number of repositories varies 15% from GHTorrent to GitHub REST API. However, we found a significant difference in the number of repositories identified as from Russia. This value decreases around 50% from GHTorrent to GitHub REST API study.

Evaluating these differences, we found that the time difference between the data collection using one API and the other can impact the result. For example, *fares-soft/terminalizer* repository, retrieved in the top-10,000 repositories using GitHub REST API, is from Jordan, but it is not present in the list retrieved from GHTorrent, probably because in the collection date it did not have the necessary number of stars to be in the top repositories. Also, another problem we found that contributes to these differences was the wrong mapping from location to a country. An example is repositories with invalid location *The Web*, which were identified to be from *Ethiopia* when using GHTorrent data, but was identified as invalid data for GitHub REST API. Also, we identified repositories from *Costa Rica* that were not identified by GHTorrent because locations such as *San Jose, CA* were wrongly mapped to *Costa Rica* instead of *USA* for GitHub REST API. These problems with the country mapping from both APIs explains the considerable difference in the number of Russian repositories for both APIs, at least 50 invalid locations that were identified as from Russia using GHTorrent.

Overall, the results from the two APIs are similar, except when analyzing the results from Russia, where we detect an issue with the country information provided by GHTorrent.

RQ2: What are the three most used languages for each country?

GitHub REST API

When analyzing the results from GitHub REST API, Table 4.2, JavaScript is the top-3 most used languages for all countries. Furthermore, it is the most used language in 16 countries. For the remaining countries, the most used programming language is Java, except for Italy that has Swift as the most used language.

GHTorrent

For the GHTorrent study, JavaScript is also in the top-3 most used language of all countries, and the most used language in 14 of them. Java is the most used language in four countries. For the two remaining countries, Italy and India, the most used language are Swift and Python, respectively.

Comparison

When comparing the results for the top-3 languages, we do not consider Finland and Norway, since they are not in the results of both APIs. For the other 19 remaining countries, we observe that the most used language is the same for 17 countries (89%). The top-3 has the same languages for 12 countries (63%), having the languages in the same order for eight of them (42%). For six other countries (32%), the top-3 from both APIs has two languages in common. The remaining country (5%) has only one language in common in both results. Therefore, the results from both APIs are very similar. Probably, the differences in the result are related to different repositories collected by the APIs, due to the time difference in data collection. Also, the fact that the number of repositories for each language does not diverge much contributes to these differences.

Country	GitHub REST API	GHTorrent
United States	JavaScript (691) Python (234) Java (163)	JavaScript (677) Python (204) Java (175)
China	Java (285) JavaScript (180) Objective-C (91)	Java (197) JavaScript (114) Objective-C (63)
United Kingdom	JavaScript (107) Java (39) Ruby (28)	JavaScript (105) Java (32) Objective-C (25)
Germany	JavaScript (69) PHP (29) Java (28)	JavaScript (54) PHP (29) Java (23)
Canada	JavaScript (71) Python (22) Go (13)	JavaScript (61) Python (15) Go (14)
France	JavaScript (50) Java (22) Python (18)	JavaScript (48) Java (18) Python (13)
Japan	JavaScript (19) Java (16) Swift (16)	Java (22) Swift (18) JavaScript (14)
India	JavaScript (25) Java (25) Python (13)	Python (14) JavaScript (13) Java (10)
Netherlands	JavaScript (28) PHP (11) Java (7)	JavaScript (27) Python (10) Objective-C (7)
Australia	JavaScript (25) Java (7) Ruby (6)	JavaScript (29) Swift (7) Ruby (7)
Sweden	JavaScript (28) Python (8) Swift (8)	JavaScript (25) Swift (7) Java (6)
Spain	Java (20) JavaScript (19) Swift (7)	Java (21) JavaScript (14) Swift (6)
Russia	JavaScript (18) Go (8) Python (8)	JavaScript (43) PHP (21) Python (10)
Italy	Swift (14) JavaScript (11) C (9)	Swift (9) JavaScript (7) Java (7)
Ukraine	Java (23) JavaScript (11) Swift (9)	Java (20) JavaScript (10) Swift (7)
Poland	JavaScript (17) Java (11) Swift (9)	JavaScript (13) Swift (12) Java (10)
Brazil	JavaScript (15) Python (5) Ruby (4)	JavaScript (9) Objective-C (4) Ruby (3)
Switzerland	JavaScript (13) Objective-C (5) PHP (3)	JavaScript (9) C (5) Objective-C (4)
Austria	JavaScript (11) Java (8) Python (7)	JavaScript (13) Python (9) Java (7)
Finland	JavaScript (18) Java (3) Python (3)	- - -
Norway	- - -	JavaScript (10) Swift (5) Java (3)

Table 4.2: Top-3 language by country using GitHub REST API and GHTorrent.

RQ3: How does the popularity of repositories vary per country?

GitHub REST API

Figure 4.1 presents the distribution of the number of stars for the projects in each studied country according to the data collected using GitHub REST API. If we focus on the median values for each country, there are no major differences. The median values range from 2,116.5 stars (Switzerland) to 2,935 stars (Russia).

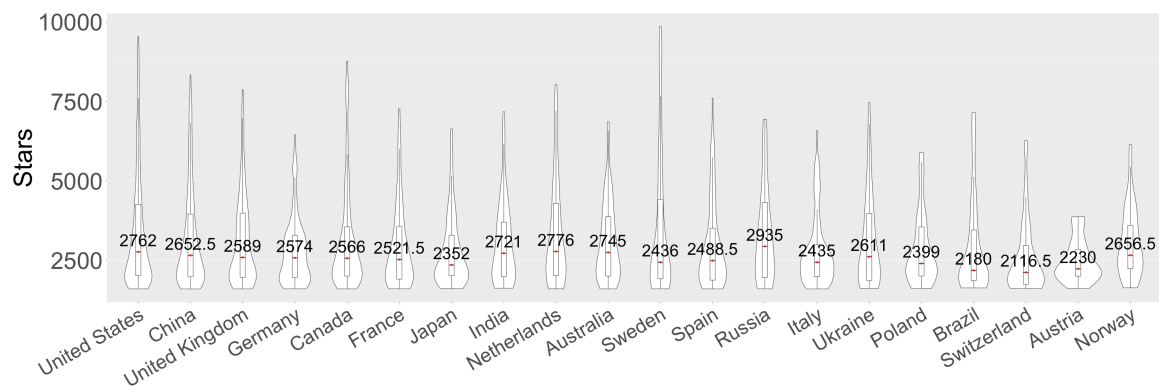


Figure 4.1: Popularity in terms of the number of stars using data from GitHub REST API

GHTorrent

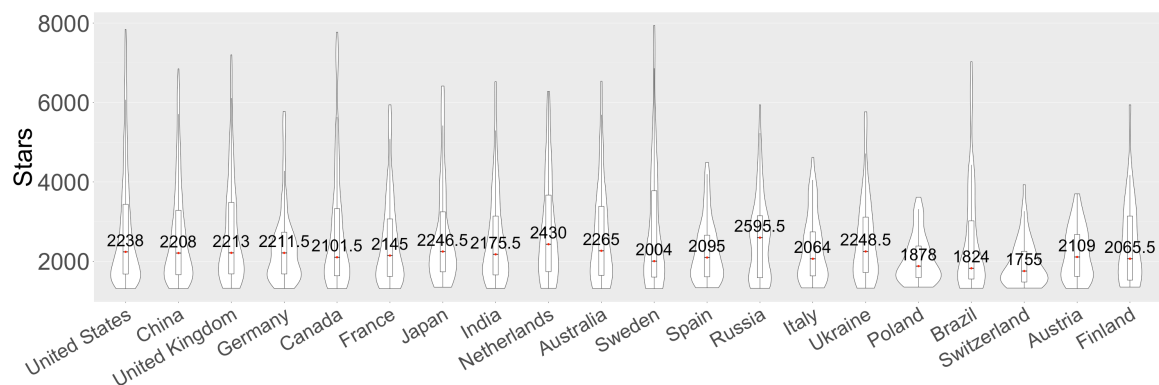


Figure 4.2: Popularity in terms of the number of stars using data from GHTorrent

For GHTorrent, Figure 4.2 presents the distribution of the number of stars for the projects in each studied country. If we focus on the median values for each country, there are no major differences. The median values range from 1,755 stars (Switzerland) to 2,595.5 stars (Russia).

4.4.0.1 Comparison

When analyzing Figure 4.1 generated from GitHub REST API data, and Figure 4.2 generated from GHTorrent data, we can see that the results are very similar. Switzerland has the lowest median value in both APIs and Russia has the highest median value for both APIs. These median values increase from GHTorrent to GitHub REST API around 4% to 28%. Also, for most countries, the graph has the same shape which means that the repositories have the same distribution regarding the number of stars for both APIs. Overall, this difference is acceptable, since GHTorrent data is from April 2018 and GitHub REST API data is from August 2018.

4.5 Limitations

For this case study, we collected data from GitHub REST API in August 2018 and used the dump from April 2018 available in Google Big Query. We tried to reproduce this study recently, but we were not able to find an up to date dump of GHTorrent in Google Big Query or on the online provider of GHTorrent, and the latest dump for download in GHTorrent platform is from June 2019. Since the process of restoring the database from the downloaded dump is time and resource consuming, we decided to maintain the results for these two data collection dates, despite the four months of difference.

4.6 Final Remarks

In this chapter, we presented a case study using data from the GitHub REST API and GHTorrent. The goal was to investigate in detail the similarities and differences of usage for both APIs. In this study, we analyzed not only the complexity of queries or setup but also the results from each API.

When comparing both APIs, the initial setup for GitHub REST API continues to be just the creation of an authentication token. For GHTorrent, different from Chapter 3, we did not download a dump and restore it to execute the queries. Instead, we used a dump available in Google Big Query, which greatly improved the initial setup and the response time. Using Google tool, we have the limitation of processing only 1TB free of charges, but this limit was not a problem because we were not processing a large volume of data. As a result, the GitHub REST API initial setup continues to be less costly than GHTorrent, because for the latter we still have to set up our Google Big Query account to access GHTorrent dumps.

The queries for both APIs had considerable complexity. For GHTorrent, the complexity was increased by the combination of data from three different tables to retrieve the desired result. The complexity of the GitHub REST API query is related to the limitation of 1,000 results per search query since we had to handle pagination and search query implementations to retrieve 10,000 repositories. Also, for every result from the *search* endpoint, we had an additional request to the *users* endpoint to retrieve the location.

The last aspect analyzed was the results. In RQ1, we retrieved the number of repositories in the top-10K from each country and constructed the top-20 countries based on the number of repositories. This raking is almost the same for both APIs with small changes in the country's position since the number of repositories from each country is very similar. Except for Russia, that had a major difference in the top-20 countries position when comparing GHTorrent and GitHub REST API results, due to a limitation in the country's inference from GHTorrent. In RQ2, we analyzed the top-3 languages for each country in the top-20 based on the most used language of each repository. The results from both APIs for the 19 shared countries are very similar, i.e, 12 countries (63%) have the same languages in the top-3, and the most used programming language is the same for 17 countries (89%). The divergence in the results is due to the slight difference in the number of repositories for each language, aligned to the time difference of data collection from both APIs. The last RQ is the most sensitive to time differences because we are analyzing the number of stars. This number is in constant change, and we can see in the result that we had a uniform growth in the number of stars when comparing the median values provided by GHTorrent and GitHub REST API. In other words, the changes in the number of stars can be explained by the difference in data acquisition, and it did not have a large impact on the result. As a result, we can conclude that the results are equivalent based on our comparative analysis of each research question.

Chapter 5

Conclusion

5.1 Overview

In this study, we analyzed and compared two widely used APIs: GitHub REST API and GHTorrent. We implemented 23 query examples based on the data collection reports from papers of two relevant Software Engineering conferences in 2017: MSR and ICSE. Based on these queries we could provide insights about initial setup, documentation quality, data availability, query complexity, and limitations for the two APIs. We concluded this master dissertation by presenting a case study that investigates open source software development in different countries. This study was implemented for both APIs, and a comparative discussion analyzed the study design, queries, and results for both APIs.

From the first study, we concluded that GitHub REST API is ahead GHTorrent in the analyzed dimensions. Besides, we found that there are some differences when comparing the results obtained using GitHub REST API and GHTorrent, most related to the time difference in the data acquisition from each API. All these results will be discussed in details in Section 5.2

5.2 Contributions

This master dissertation contributes to the following aspects:

- **Real query examples.** We identified data collected using GitHub REST API and GHTorrent for papers from two major Software Engineering conferences. Based on these real scenarios, we implemented example queries to collect the desired data and discussed them in detail.

- **API analysis.** Based on the effort spent on the queries implementation, we provide some sights about different dimensions of each API:
 - **Initial Setup.** In this aspect, we concluded that the complexity is higher for GHTorrent. When evaluating GitHub REST API, the only required initial setup is to retrieve an authentication token to increase the number of requests that can be made. On the other hand, to access GHTorrent data it is necessary to download and store a large amount of data, which is not a viable option in some cases. Thus, if the goal is to retrieve a small set of data, it is preferable to use GitHub REST API or some online provider of GHTorrent data.
 - **Documentation.** Both APIs provide documentation about the data they provide and the methods to access it. However, Github REST API is more complete and updated when compared to GHTorrent. Regardless, the basic necessary information is available in the documentation of both APIs.
 - **Data availability.** GitHub REST API is the official API provided by GitHub to access its public data. Although GHTorrent aims to be an offline mirror of GitHub, not all data available through GitHub REST API is available on GHTorrent. By contrast, it is important to highlight that GHTorrent provides access to historical data, which is not possible through GitHub REST API. For all these reasons, it is important to analyze the necessary data to be collected to decide which API is more suitable.
 - **Query Complexity.** The query complexity was analyzed regarding different aspects from GHTorrent and GitHub REST API. For GHTorrent, we considered the number of tables and joins, and for GitHub REST API we verified the number of endpoints and requests. We concluded the complexity for GHTorrent is a bit higher than for GitHub REST API.
 - **Limitations.** Both APIs have limitations, but regarding different aspects. For example, GitHub REST API has a limitation in the number of results that can be returned by the search endpoint. On the other hand, GHTorrent has a limitation in the data available in their dumps. In the end, we could not find a workaround for all limitations; therefore, it important to understand them before start using these APIs.
- **Results analysis.** Based on the case study we provided about open source software development in different countries, we were able to compare both APIs regarding the results they provide. We found some differences, but most of them

related to the time difference in data acquisition. Overall, the results achieved by collecting data from both APIs are very similar. This fact implies that the results of the study should not be heavily affected by the chosen API.

5.3 Related Work

GitHub data has been widely used in Software Engineering Research; as a result, it is important to investigate the reliability of this data and the potential threats that might appear when using the data mined from GitHub. Kalliamvakou et al. [2014] identified two benefits and nine potential threats related to GitHub data. Moreover, the authors proposed a set of recommendations to avoid threats. For example, a peril is that two thirds of projects from GitHub are personal, so to overcome this limitation we should consider the number of contributors when retrieving projects from GitHub. This work differs from ours because we are not analyzing the quality of the data collected from GitHub. Instead, we are analyzing and comparing two APIs that provide access to that data.

Another study investigates how the GHTorrent dataset is used on empirical studies and how much data is consumed [Borges et al., 2016a]. For this study, the authors collected papers that cite two studies about GHTorrent and selected only the ones that were collecting data. Based on the analysis of these papers, the authors concluded that the data related to repositories and users are the most used data from GHTorrent. Regarding the amount of data consumed, the number of queried repository data varies from 7 to 34,6 million, and for users this number ranges from 150 to 13.2 million.

Cosentino et al. [2016] studied how researchers mine GitHub, analyzing the empirical methods employed, used datasets, and reported limitations. For that analysis, they collected papers containing *GitHub* (or variations) in the title, abstract, author keywords or index terms, and also checked their related work to find more papers. From these papers, the author only selected the ones that indeed use GitHub data. Based on these papers, the authors concluded that the most used empirical method employed was direct observation of GitHub metadata. Also, they found that in most cases the data was collected using GitHub REST API followed by GHTorrent. For limitations regarding data collection, the limit in the number of requests was mentioned by papers using GitHub REST API. In the case of GHTorrent, the limitations are related to dataset size and data freshness.

Even though we have some papers about GitHub REST API and GHTorrent, to the best of our knowledge, this is the first work that compares both of them, using real

world examples and a case study.

5.4 Future Work

In this dissertation, we performed a comparison of two APIs used to collect data from GitHub: GitHub REST API and GHTorrent. We plan as an extension of this work:

- **GitHub GraphQL API.** GitHub provides now a new API to collect data: GraphQL API. We did not discuss this API in this study, therefore a suggestion for future work is to include this new API in the study.
- **Case Study.** We provided a case study that uses data about repositories. As a result, we could only analyze the results related to repository data. In this way, as future work, we suggest to analyze the results for other entities from GitHub, for example: pull requests, issues, and commits.

Bibliography

- Al Omran, F. N. A. and Treude, C. (2017). Choosing an nlp library for analyzing software documentation: a systematic literature review and a series of experiments. In *14th International Conference on Mining Software Repositories (MSR)*, pages 187--197.
- Beller, M., Gousios, G., and Zaidman, A. (2017). Oops, my tests broke the build: An explorative analysis of travis ci with github. In *14th International Conference on Mining Software Repositories (MSR)*, pages 356--367.
- Borges, H., Coelho, J., Carvalho, P., Fernandes, M., and Valente, M. T. (2016a). Como pesquisadores usam o dataset GHTorrent? In *5th Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM)*, pages 1--8.
- Borges, H., Hora, A., and Valente, M. T. (2016b). Understanding the factors that impact the popularity of GitHub repositories. In *32nd International Conference on Software Maintenance and Evolution (ICSME)*, pages 334--344.
- Cosentino, V., Izquierdo, J. L. C., and Cabot, J. (2016). Findings from GitHub: methods, datasets and limitations. In *13th International Conference on Mining Software Repositories (MSR)*, pages 137--141.
- Cosentino, V., Izquierdo, J. L. C., and Cabot, J. (2017). A systematic mapping study of software development with GitHub. *IEEE Access*, 5:7173--7192.
- Floyd, B., Santander, T., and Weimer, W. (2017). Decoding the representation of code in the brain: an fmri study of code review and expertise. In *39th International Conference on Software Engineering (ICSE)*, pages 175--186.
- Gharehyazie, M., Ray, B., and Filkov, V. (2017a). Some from here, some from there: Cross-project code reuse in github. In *14th International Conference on Mining Software Repositories (MSR)*, pages 291--301.

- Gharehyazie, M., Ray, B., and Filkov, V. (2017b). Some from here, some from there: Cross-project code reuse in github. In *14th International Conference on Mining Software Repositories (MSR)*, pages 291--301.
- Gousios, G. and Spinellis, D. (2012). GHTorrent: GitHub's data from a firehose. pages 12--21. ISSN 2160-1852.
- Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., and Damian, D. (2014). The promises and perils of mining GitHub. In *11th International Conference on Mining Software Repositories (MSR)*, pages 92--101.
- Kikas, R., Gousios, G., Dumas, M., and Pfahl, D. (2017). Structure and evolution of package dependency networks. In *14th International Conference on Mining Software Repositories (MSR)*, pages 102--112.
- Ma, W., Chen, L., Zhang, X., Zhou, Y., and Xu, B. (2017). How do developers fix cross-project correlated bugs? a case study on the github scientific python ecosystem. In *39th International Conference on Software Engineering (ICSE)*, pages 381--392.
- Macho, C., McIntosh, S., and Pinzger, M. (2017). Extracting build changes with builddiff. In *14th International Conference on Mining Software Repositories (MSR)*, pages 368--378.
- Osman, H., Chis, A., Corrodi, C., Ghafari, M., and Nierstrasz, O. (2017). Exception evolution in long-lived java systems. In *14th International Conference on Mining Software Repositories (MSR)*, pages 302--311.
- Wan, Z., Lo, D., Xia, X., and Cai, L. (2017). Bug characteristics in blockchain systems: a large-scale empirical study. In *14th International Conference on Mining Software Repositories (MSR)*, pages 413--424.
- Xiong, Y., Wang, J., Yan, R., Zhang, J., Han, S., Huang, G., and Zhang, L. (2017). Precise condition synthesis for program repair. In *39th International Conference on Software Engineering (ICSE)*, pages 416--426.
- Yang, D., Martins, P., Saini, V., and Lopes, C. (2017). Stack overflow in github: any snippets there? In *14th International Conference on Mining Software Repositories (MSR)*, pages 280--290.
- Zampetti, F., Scalabrino, S., Oliveto, R., Canfora, G., and Di Penta, M. (2017). How open source projects use static code analysis tools in continuous integration pipelines.

In *14th International Conference on Mining Software Repositories (MSR)*, pages 334-344.