Extracting software development information from FLOSS Projects in GitHub

Miguel Ángel Fernández, Gregorio Robles
GSyC/LibreSoft
Universidad Rey Juan Carlos
{ma.fernandezsa@alumnos., gregorio.robles@}urjc.es

Abstract

GitHub is the most used online code platform in the world, with +58 million repositories. Mining information from these millions of projects and analysing that data is very useful for both researchers and companies. This paper presents a methodology for extracting information from these Free/Libre/Open Source Software repositories stored on GitHub, applied to a case study: the search of UML models for quantify and analise its use in this type of projects. For that, it starts from a database provided by the GHTorrent project (which offers an offline mirror of data from the GitHub REST API), and a series of scripts are used for extracting metadata from the repositories in order to look for patterns and/or specific file extensions. Once the interesting projects have been identified through an external process, they are analysed with Perceval, a program which extracts metrics from them.

1 Introduction

With more than 21 million users and more than 58 million repositories[1], GitHub is the most used online code platform in the world. We can extract huge amounts of information from these million projects about how is the software being developed: productivity, issue-tracking methods, developer-to-developer relations, etc. [RGBM06] For researchers, GitHub is an endless source of publicly available data for potential studies, and for companies (specially for big ones) is probably the best way to know deeply how its software is evolving over time, so they can use this data to make the right decissions for the future (known as data-driven decissions).

It is widely known that GitHub allows to filter projects by a certain programming language (e.g. Python, C++, etc.), but GitHub does not offer a mechanism through which you can identify projects containing files with a certain extension or how many files match with some word or pattern in their filename in a repository. Therefore, the main research questions are:

- RQ1: How many GitHub repositories contains at least one file with a certain extension or a certain pattern in its filename?
- RQ2: What's the history of those files in the lifespan of the project?

Thus far, many studies focused in one single project or in a limited dataset when software development is analysed. Our goal is to extract and analyse data from all GitHub in a scalable, semi-automated way in order to obtain information about the usage of a certain file type, programming language or/and any search that can be expressed into patterns and heuristics.

To analyse the data from all GitHub correctly, we needed a static, reliable and updated source of this platform. This source is provided by the GHTorrent project [Gou13], whose purpose is "to build a scalable,

Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution SATToSE 2017 (sattose.org).
07-09 June 2017, Madrid, Spain.

[1] `https://github.com/about`

queriable, offline database with all the information provided by GitHub REST API"[2].
The dataset used in this study is from February, 2016. At that time, there were a total of 26 million repositories (approx.).
As the main research that motivated this study was interested in identify UML models introduced in a project by its owners, we had to filter those projects to discard forked repositories. Applying that filter we got 12,847,555 repositories ready to be analysed.
GHTorrent does not provide information about files from a given repository, so it's necessary to retrieve this information from GitHub through its API.

2 Methodology

Our methodology is based on five main stages:
In a preliminar stage, we retrieve the dataset from GHTorrent in order to obtain the whole list of repositories available in GitHub from a particular dump. From this list we can filter projects if necessary: non-forked ones, most used language, etc.
As the file information of the repositories is missed in this offline database, it's mandatory to obtain this data through the GitHub API only for the selected projects.

Once the list of repositories is ready, the first step is to download the file list from each repository in the filtered projects list (See Fig. 1). For making this, our first script sends authenticated `http` requests to GitHub API (using a given token from a real GitHub account). These requests ask for the main branch of the repository (master); if this branch is not found, another request is sent asking for the default branch. Once the main branch is obtained, another request is sent to get the file list (trees) of the repository.

The second stage is to identify the files and/or patterns of interest by iterating over all file lists from each repository, checking whether any file matches with our search parameters (See Fig. 2). When a match is found, the `url` of the repository is added to a `url` list, that will be the list of "potential" (not validated) projects. Additionally, an intermediate stage may be required between this phase and the next one. This will be detailed in "Case of Study" section.

The third step consists on analysing those repositories with a tool called `Perceval`, an evolution from `CVSanalY`[RGBICH09] software, included in GrimoireLab[3], a free-software toolset developed by

[2] `http://ghtorrent.org/`
[3] `http://grimoirelab.github.io/`

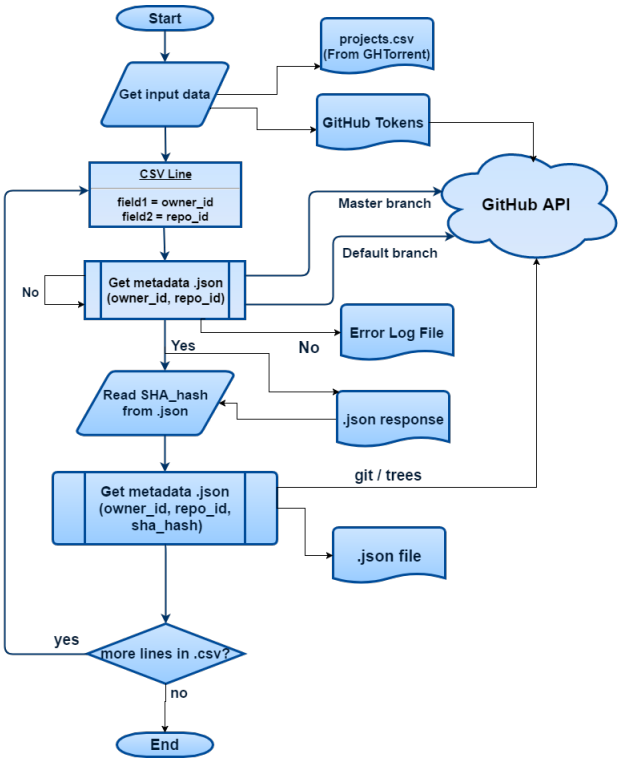

Figure 1: Detailed schema from stage 1 script.

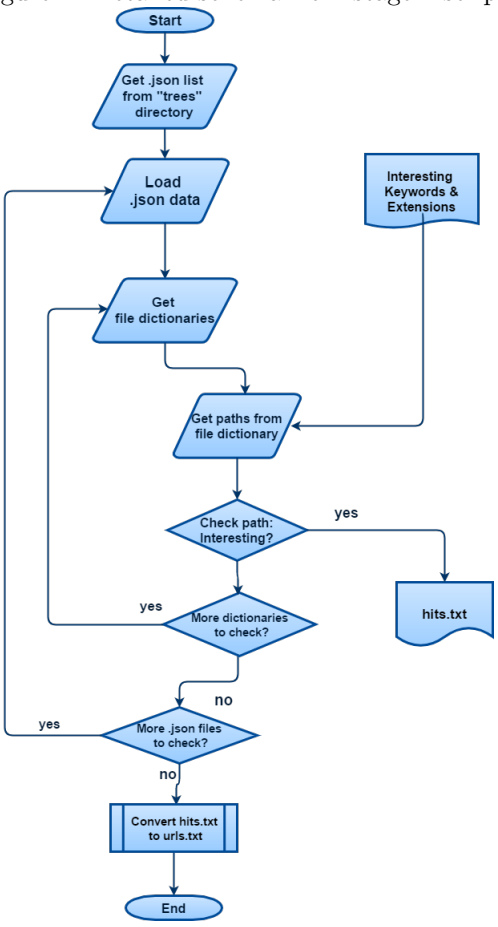

Figure 2: Detailed schema from stage 2 script.

Bitergia[4]. Given a repository, `Perceval` clones it locally and extracts metadata. This metadata includes information like number of commits, contributors, location, company, etc. The last phase is based on building a database with the files produced by `Perceval` (one file per repository), so the obtained data can be easily queriable.

Limitations: The aim for this tool to be scalable implies to deal with several impediments. GitHub API has a limitation of 5,000 requests per hour & account. As it is explained before, a maximum of three requests are made to obtain the main branch for each repository, as we are not interested in secondary branches for our case of study. Making the calculations about how much time would it take to complete this first stage, we calculated that 322 days would be needed, so we opted for parallelizing the process using many different GitHub accounts. Finally, it took almost 90 days to complete this task.

In addition to the GitHub API limitation, we may encounter technical limitations: it is desirable to have a fast and stable Internet connection, enough space in the hard drive and also use a powerful computer, as:

- The size of the whole set of partial output files (compressed) is 140 GB.
- Each of these partial files has to be opened and analysed.
- At the analysis stage, the interesting projects are cloned (downloaded + uncompressed).

3 Case study

This methodology was applied for the first time to the study: "The quest for Open Source projects that use UML: Mining GitHub" [HQC+16]. This research has been a collaborative work between Chalmers University and Rey Juan Carlos University, led by Michel Chaudron, Gregorio Robles, Truong Ho Quang and Regina Hebig.
The main goal of this study is to deepen in the knowledge of usage and evolution of UML models in Free/Libre/Open Source Software (FLOSS) projects, tracking them throughout the whole projects life-span. Some of the research questions of this study were:

- RQ1: Are there GitHub projects that use UML? Which are these projects?
- RQ2: Are there GitHub projects in which the UML models are also updated?

[4] `https://bitergia.com/`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
      <xs:include schemaLocation="DataTypes.xsd"/>
      <xs:element name="shippingOrder">
          <xs:complexType>
             <xs:sequence>
                <xs:element name="shippingId"type="int"/>
                <xs:element name="origin" type="Origin"/>
                <xs:element name="destination" type="Destination"/>
                <xs:element name="order" type="Order"/>
             </xs:sequence>
          </xs:complexType>
      </xs:element>
</xs:schema>
```

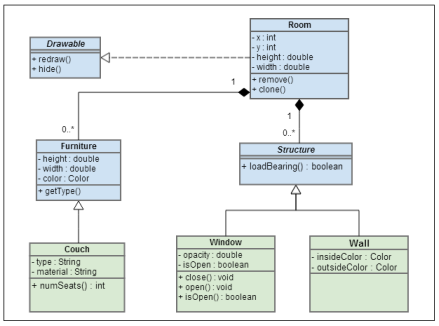

Figure 3: Examples of how UML models can be found: XML-alike (above) or image type (below).

- RQ3: When in the project are new UML models introduced?

This case of study description is focused on the implementation of the technical part of the study, showing some of its data and results.

The first step specified in the methodology is, by default, independent from the case of study. The specific part is located mainly on the second phase, where the search patters are detailed. It is important to know if there is some pecularity in the data we are looking for, for example, UML models can be found in many different formats, but they can be classified into two main types: text-based models, which are XML-alike files and image-based models (See Fig. 3). This entails an additional problem as an external validation of the data is required.

In this case, we are interested in looking for files whose extension matches with all possible extensions which an UML model can be found: `"uml"`, `"xmi"`, `"uxf"` and `"xdr"` (most common extensions) and files whose name may include one of the following keywords: `"xmi"`, `"uml"`, `"diagram"`, `"architecture"` and `"design"` in addition to have one of these extensions: `"xml"`, `"bmp"`, `"jpg"`, `"jpeg"`, `"gif"`, `"png"` and `"svg"`.

As described before, an additional problem arose: from all the identified files with the interesting extensions, how many of those files really are a UML model, and how many are not? This problem was solved by Chalmers team using heuristics for the XML-alike files and image processing and machine learning techniques for image files [HQCS+14a, HQCS+14b]. Once the repositories were verified, they were analysed using Perceval and their data were added to the database.

3.1 Results

A total of 24,797 repositories have at least one UML file, and a total of 93,648 UML files were identified. That means that in all GitHub only 0.193% of the repositories contains at least one UML model. As the main aim for this study was studying projets that are interesting from an industry perspective, it was necessary to filter those projects which are not short-term and do not consist of a single contributor.
Short-time projects were defined as those repositories that:

1. Active time (time span between first and last commit) less that 6 months
2. Less than 2 contributors
3. Less than 10 commits

After this filtering, the final set contained 4,650 UML-projects (out of 24,125).

4 Future work

One major issue is related to GHTorrent, as `csv` files do not maintain regularity in their format, so each new version from the database has to be manually revised in order to adapt the input data. This should be improved for future updates of the current dataset. The enhancement of the paralellization for the retrieval is another goal, as is a huge time-consuming stage.

Acknowledgments

This work has been funded in part by the Spanish Gov. under SobreSale and by the Comunidad de Madrid under "eMadrid - Investigación y Desarrollo de tecnologías para el e-learning en la Comunidad de Madrid".

References

[Gou13] Georgios Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[HQC+16] Regina Hebig, Truong Ho Quang, Michel R. V. Chaudron, Gregorio Robles, and Miguel Angel Fernandez. The quest for Open Source projects that use UML: Mining GitHub. In *Proceedings 19th International Conference on Model Driven Engineering Languages and Systems*, pages 173–183, 2016.

[HQCS+14a] Truong Ho-Quang, Michel R. V. Chaudron, Ingimar Samúelsson, Jóel Hjaltason, Bilal Karasneh, and Hafeez Osman. Automatic classification of UML class diagrams from images. In *Proceedings of the 2014 21st Asia-Pacific Software Engineering Conference - Volume 01*, pages 399–406, 2014.

[HQCS+14b] Truong Ho-Quang, Michel R. V. Chaudron, Ingimar Samúelsson, Jóel Hjaltason, Bilal Karasneh, and Hafeez Osman. Automatic classification of uml class diagrams from images. In *Proceedings of the 2014 21st Asia-Pacific Software Engineering Conference - Volume 01*, APSEC '14, pages 399–406, Washington, DC, USA, 2014. IEEE Computer Society.

[RGBICH09] Gregorio Robles, Jesús M González-Barahona, Daniel Izquierdo-Cortazar, and Israel Herraiz. Tools for the study of the usual data sources found in libre software projects. *International Journal of Open Source Software and Processes*, 1(1):24–45, 2009.

[RGBM06] Gregorio Robles, Jesus M Gonzalez-Barahona, and Juan Julian Merelo. Beyond source code: the importance of other artifacts in software development (a case study). *Journal of Systems and Software*, 79(9):1233–1248, 2006.