UNIVERSIDAD
REY JUAN CARLOS

GRADO INGENIERÍA SISTEMAS AUDIOVISUALES Y MULTIMEDIA

Curso Académico 2017/2018

Trabajo Fin de Grado

HERRAMIENTA PARA IDENTIFICAR Y EXTRAER MASIVAMENTE ARTEFACTOS DE GITHUB

Autor : Miguel Ángel Fernández Sánchez

Tutor : Dr. Gregorio Robles

Trabajo Fin de Grado

Herramienta para Identificar y Extraer Masivamente Artefactos de GitHub

Autor : Miguel Ángel Fernández Sánchez

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Trabajo Fin de Grado se realizó el día de de 2018, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2018

Dedications

A mi padre y a mi madre:
Gracias por vuestro
incansable esfuerzo y apoyo.
Este es vuestro éxito.

To my mother and father:
Thank you for your
endless effort and support.
This is your success.

II

Acknowledgements

First, I want to dedicate these first words to my parents and the rest of my family. I am grateful for the values they taught me about persistence, tolerance and self-sufficiency. Their constant effort and support made possible for me studying at the university, which is something I feel really proud of.

I want to thank my tutor Gregorio, for granting me the great opportunity of working in *GSyC/LibreSoft* research group where I have lived some of my great experiences at the university; growing as a person, meeting wonderful people, travelling and learning far beyond I could have ever imagined.

To my friends and classmates from the university: Eduardo, Eva, Diego, Sara, Raúl, Olalla, Javi, Bea, Dani, Carol, Marta, Cristina, Nuria, Raquel and many more. Without your friendship, help and support I would have not reached this goal, thank you.

To the people who have helped me to review and improve this thesis: Gregorio, Santiago, Valerio, Fernando, Saray, Eva and Sara. Thank you for your patience and sharing your time and knowledge so generously.

Finally, thanks to the people who contributed with their GitHub account to make this study possible: Gregorio R., Michel C., Truong H.Q., Eva H., Olalla S., Beatriz C., Diego S., Eduardo D., Javier M., Diego J., Jose M., Alejandro F., Ángel C., Alejandro, Cristina, Fernando, Irene, Nacho, Ana R. and Ronny.

Summary

In this thesis, a method for extracting and analyzing information from Free / Libre / Open-source software (FLOSS) repositories hosted in *GitHub*, a popular software development platform with millions of projects, is presented. Thus far, many studies focused in one single project or in a limited dataset when software development is analyzed. The goal of this project is to extract and analyze data from the whole GitHub platform in a scalable, semi-automated way in order to obtain information about the usage of a certain file types, programming languages or/and any search that can be expressed into patterns and heuristics.

To perform the data extraction, the tool starts with a dataset provided by the *GHTorrent* project, a database which contains a partial mirror of data from the *GitHub API*. Then, a series of scripts are used for extracting file-related data from the repositories, identifying interesting projects looking for patterns and/or specific file extensions and analyzing those positive results using an external tool called *Perceval*, which extracts additional metrics from them. Finally, this information is structured into a database where a deeper analysis can be performed.

This project was created to be applied to a research case study: the search of *UML* models in FLOSS projects, although it has been used succesfully to identify software architecture documents as well. With the identification of such artifacts, the knowledge of usage and evolution of UML models can be widened, quantifying and analyzing its use in projects hosted in the GitHub platform, tracking them throughout the whole projects life-span.

Resumen

En esta memoria se presenta un método para extraer y analizar información de repositorios de software libre y *open-source* (FLOSS) alojados en *GitHub*, una plataforma dedicada al desarrollo de software con millones de proyectos software. En el pasado se habían realizado varios estudios analizando el desarrollo de software, concentrándose en un conjunto muy limitado de proyectos para realizar dicho análisis. El objetivo de este proyecto es extraer y analizar datos obtenidos de todo GitHub de una forma escalable y semi-automática, con el fin de recopilar información sobre el uso de cierto tipo de ficheros, lenguaje de programación y/o cualquier búsqueda que pueda ser expresada en forma de patrones y heurísticos.

Para realizar la extracción de los datos, la herramienta usa un conjunto de datos iniciales proporcionados por el proyecto *GHTorrent*, una base de datos que contiene una copia de datos obtenidos de la *API* de GitHub. A continuación, se usan una serie de *scripts* para extraer datos relacionados con los ficheros que contienen los repositorios, identificar aquellos proyectos de interés a través de patrones y/o extensiones de fichero específicas y por último analizar aquellos resultados positivos usando una herramienta externa llamada Perceval, que extrae métricas adicionales de dichos resultados. Finalmente, esta información es estructurada en una base de datos donde se puede realizar un análisis más profundo.

Este proyecto fue creado para ser aplicado a un caso de estudio: la búsqueda de modelos *UML* en projectos FLOSS, así como de archivos que describen arquitecturas software. La herramienta permite, por tanto, ayudar en el desarrollo de estudios que profundicen en el conocimiento sobre el uso y la evolución de los modelos UML, cuantificando y analizando dicho uso en proyectos alojados en la plataforma GitHub y rastreando su evolución a través de todo el tiempo de vida de cada proyecto.

Contents

List of Figures

Chapter 1

Introduction

GitHub is the most used online code platform in the world. We can extract huge amounts of information from millions of projects about how software is being developed: productivity, issue-tracking methods, developer-to-developer relations, etc. [12] For researchers, GitHub is an endless source of publicly available data for potential studies, and for companies and communities it is probably the best way to know how (its) software is evolving over time, so they can use this data to make the right decisions for the future (known as data-driven decisions).

I was working as a researcher assistant in the *GSyC/LibreSoft*[1] research group at the Universidad Rey Juan Carlos when I started to deepen into Free/Libre software and the world of metrics. There, I had the great luck of start participating in a study that Dr. Robles was doing with researchers from Chalmers University in Gothenburg, Sweden. The main aim of their research was to learn how *UML* models[2] are used in public *FLOSS*[3] projects (particularly, projects hosted in GitHub), as most of the previous studies focused only on its industrial use.

Given the large size of GitHub as a platform and its access limitations [9], this research required to build a tool which extracts Git data from all the existing repositories on GitHub, then filters those repositories which may include at least one *UML* model and finally analyzes the set of projects that meet this condition more in-depth.

[1] `http://www.libresoft.es`

[2] `http://uml.org/what-is-uml.htm`

[3] Free/Libre and Open Source Software. Complete definition at section 1.2

Since early stages of this research, this tool was designed and built using a modular structure, thought to be reusable and adaptable to any kind of files that have to be found, but also to the way GitHub provides data, with the hope to be useful for other researches or any other person who might be interested on extracting this information. Besides, the scale of this project presented a personal challenge which provided me the opportunity to extend my knowledge and obtain a valuable point of view about research and large-scope projects.

1.1 Context

The way people develop software has evolved during the last years [18]. There is an extended social perception of developers to being introverted, solitary people, with very few social skills, but this sense is outdated: the paradigm has changed, as coding has become a social activity. Developers who work collaborating with others produce better software [20, 10], as they are continuously reading code from other developers, getting feedback about their contributions and consequently widening their knowledge with new procedures and ideas.

In the last few years we have seen how this social trend came to software with the emergence of social coding platforms, such as GitHub[4], BitBucket[5], GitLab[6], etc.

Social interactions in these platforms are about how software is developed and maintained but also how contributions are managed (code review, continuous integration, etc.).

So, for example, on GitHub[7] we can interact to obtain or produce different data:

- *Forking* a repository (create an editable copy of a project).
- Viewing the changes in the last *commit* (i.e., the difference between most recent version and last version of the project).
- Contributing to a repository.
- Submitting a *Pull Request* (proposal to merge changes in a project).

[4] `https://github.com`
[5] `https://bitbucket.org/product`
[6] `https://gitlab.com/`
[7] `https://github.com`

These interactions leave traces which remain stored in some computer on Earth and generally are visible to everybody. This means we can obtain these data (though not all of it is publicly available) through the Internet using an *API*[8]. Once this information is retrieved, we can analyze it and extract conclusions about it.

To complete the context for this thesis, it is necessary to situate what UML is and why it is important. The Universal Modeling Language (UML) is a graphic, descriptive language to visualize, specify, construct and document the artifacts from a system (mainly: structure, behavior and interaction) containing a great amount of software [14]. UML provides a standard way to write the plans of a system, covering from conceptual aspects like business processes and system features to particular aspects like classes in a certain programming language, database schemas and reusable software components with a higher level of abstraction. Therefore, modeling with UML is considered as one more component in a software development process [1].

For industrial software, the use of UML is commonly accepted and software engineering researchers in the area of modeling have made efforts to collect examples of projects that use modeling and their models. Nonetheless, industrial projects are very averse to share their models because of these projects' licenses and privacy issues (e.g., its current state, new features, among others). This situation hinders the task of creating a models dataset for researchers. So, in Summer 2016 a dataset containing around 81 models from open source projects was probably the one with the larger number of models that could be accessed. At that moment, there was not much research about the use of UML in open source. Furthermore, most open source code platforms do not provide funcionality for model management.

In this context, a tandem research was born, merging the experience of LibreSoft team researchers (from the Universidad Rey Juan Carlos) about FLOSS and software development processes and the experience of the Software Engineering Division researchers (at Chalmers University) about software modeling to obtain quantitative data about the use of UML in open source projects. To make this study a reality, they decided to get these projects from GitHub, because it is the code platform containing the greatest amount of FLOSS projects. This motivated to create the tool presented in this thesis, that had as aim to perform a systematic search looking for UML models in an automated and scalable way.

[8] See the definition for API in section A.2

As I explain in the "Results" section (5.1), a total of 93,596 publicly available UML models have been identified thanks to this research. This amount of models compounds a dataset that is at least two orders of magnitude larger than the previous UML datasets from previous studies. The results obtained with this tool and its subsequent analysis led to several scientific papers, where I am a co-author, published in top international Software Engineering conferences, as I detail on section 5. The published papers are available in Appendix C.

1.2 Free/Libre/Open-Source Software

The tool presented in this thesis is distributed as Free/Libre software. Moreover, in the following sections we are going to talk about software and its development process, focusing on Free/Libre and Open-source software (*FLOSS*) projects. Therefore it is inevitable to describe briefly what FLOSS is about.

According to the **Free Software Foundation**[9]:

> *Free software* means software that respects users' freedom and community. Roughly, it means that the users have the freedom to run, copy, distribute, study, change and improve the software. Thus, free software is a matter of liberty, not price. (...) (For an extended definition, see section A.3).

Due to the connotations which the word *free* has in English language, there is a definition for *Open source* software by the **Open Source Initiative**[10] which slightly varies from the *FSF* definition. About this difference, Richard Stallman (founder of the *FSF*) comments:

> The term *open source* software is used by some people to mean more or less the same category as free software. It is not exactly the same class of software: they accept some licenses that we consider too restrictive, and there are free software licenses they have not accepted. However, the differences in extension of the category are small: nearly all free software is open source, and nearly all open source

[9] https://www.gnu.org/philosophy/free-sw.html
[10] https://opensource.org/osd

software is free.

During my degree, I was fortunate to be taught within *FLOSS* values, as most of my professors plead for Free/Libre software as their teaching model, by which I have obtained a distinctive knowledge and perspective as a professional but also as an individual.

1.3 Structure of the thesis

In this thesis, the main objectives which were marked to build the tool are presented in Chapter 2 "Objectives", followed by a brief explanation of the technology which have been used to achieve these objectives in Chapter 3 "State of the art". Next, in Chapter 4 "Design and implementation" the design process and the tool architecture is detailed, itemizing on each one of its components. Thenceforth, the performance and results of the tool are explored in Chapter 5 "Results" by means of two different use cases, presenting the obtained results but also taking into account the technical particularities which arose during its development and how they were solved. Furthermore, some conclusions are presented in Chapter 6 "Conclusions", reviewing whether the marked objectives have been accomplished or not, along with a brief discussion about the limitations of the tool and future work.

Additionally, this thesis contains the following annexes:

A. Definitions

B. Code of the tool

C. Published papers

Chapter 2

Objectives

This tool was created with the purpose to address the following two research questions:

- RQ1: How many GitHub repositories contain at least one file with a certain extension or a certain pattern in its file-name?
- RQ2: What is the history of those files in the life-span of the project?

Thus far, many studies focused on one single project or in a limited dataset, whereas the aim of this research has been to obtain the whole dataset of repositories hosted in the GitHub platform in order to obtain quantitative information about the usage of a certain file type, programming language or/and any search that can be expressed into patterns and heuristics. Then, we store that data in a database where it can be analyzed in a deeper way to extract elaborate information about the data-set.

To meet these requirements, the tool had to achieve the following main objectives:

- OB1: Extract the whole set of projects hosted in GitHub.
- OB2: Collect data from all public projects about their file structure in a scalable, automated way.
- OB3: Establish a procedure to filter this extracted data using a determined kind of heuristics and patterns.
- OB4: Analyze positive GitHub repositories (with at least one positive result) to obtain

enhanced Git data from them.

- OB5: Store this enhanced data in a database whose structure allows to query information of interest.

To accomplish OB1 in a proper way we needed a static, reliable and updated source of GitHub. GitHub allows to obtain its information through its *API*, but this entails several limitations. To mention the most important ones:

- The GitHub API has a maximum rate of 5,000 requests per hour with each account.
- GitHub is not static (e.g., new repositories are added, existing ones are modified) which leads to the problem of accessing concurrently to its data.
- There is no direct way of obtaining the amount of repositories in the platform at a certain date using its *API*.

These limitations led to the decision of using the *GHTorrent* project [4], which offers a queriable, offline *MySQL* database with most of the information which the GitHub API can provide.

GitHub allows to filter projects by a certain programming language (e.g., *Python*, *C++*), and offers 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 file-name in a repository. But this filtering is limited to 10,000 projects, which is a relative small number if we consider that it hosts millions of them. Therefore, to achieve OB2 and OB3 we had to extract the list of files from the main branch[1] of each repository. However, *GHTorrent* data does not contain file-related information, so it is necessary to define a method to access the GitHub API in order to retrieve this information so later it can be applied a set of filters to determine which repositories meet the established requirements. Once OB1, OB2 and OB3 are achieved, the obtained results will constitute the answer to RQ1.

OB4 and OB5 have to be completed to answer *RQ2*. Analyzing Git repositories is not a trivial task, as it is crucial to retrieve complete and reliable information from them to perform a robust analysis. That is why we decided to use a tool called *Perceval* [3] which belongs to

[1] Definition of branch and other Git objects in section A.1

the *GrimoireLab* project, an evolution from *CVSanalY* [11] software. *GrimoireLab* is a mature and Free/Libre tool-set for software development analytics, used to extract metrics from large, industrial projects but also to perform research studies in academia. With such background, *Perceval* tool is suitable to comply with OB4. Nonetheless, it should be noted that this tool is not valid to perform OB2 and that's why the direct access to the GitHub API is still needed. Lastly, to accomplish OB5 the approach is to store the data obtained with *Perceval* in a database to execute the necessary queries to obtain answers for RQ2 and any further questions.

Chapter 3

State of the art

There are multiple tools to analyze GitHub repositories. Nonetheless, at the moment when this tool was designed there was no application capable of meeting the diverse requirements for the researcher's needs to complete this study. The uniqueness of the structure of this information along the process and its scale, together with the advantage of controlling every stage of the tool in order to ensure the truthfulness of the results, triggered the need of creating a customized, modular tool to satisfy every aspect of the study as much as possible.

These are the technologies that were chosen to build this tool. Every technology will be motivated in Chapter 4.

3.1 Python

Python[1] is an interpreted, object-oriented, high-level, open source programming language for general-purpose programming created by Guido van Rossum in 1991 [19]. Nowadays, the most recent version is 3.6.4, from December, 2017. Its design is focused on code readability and clear syntax, making possible to program using fewer lines of code comparing to other programming languages as *C++* or *Ada*.

Python features a large standard library, which includes many different tasks from text pattern

[1] `https://www.python.org/`

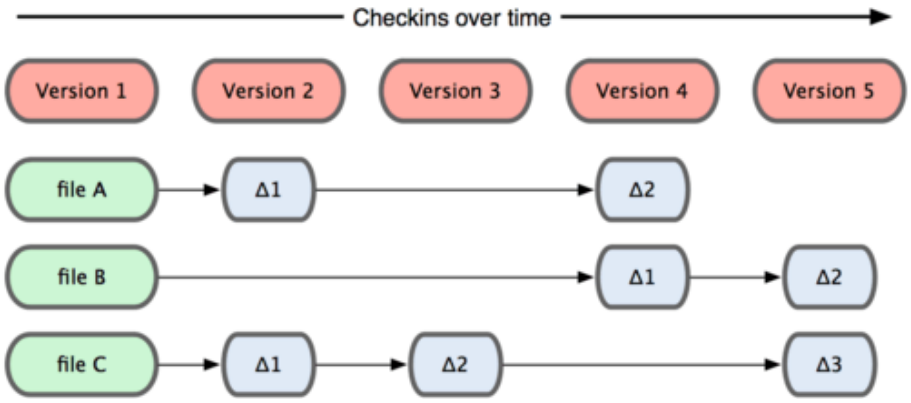


Figure 3.1: How other non-*git VCS* store information [17].

matching to network scripting, in addition to a vast collection of third-party application libraries. Other remarkable features are portability, as *Python* interpreters are available for many operating systems; and the component integration, as *Python* scripts can easily communicate with other parts of an application or code, like *C++* libraries, *MySQL* databases, etc.

3.2 Git

Git is an open-source Version Control System (*VCS*), originally developed in 2005 by Linus Torvalds [16]. As any *VCS*, *git* is a system that records changes to a file or set of files over time so that you can recall specific versions later. According to last surveys, it is by far the most used *VCS* in the world[2].

The major difference between *git* and any other VCS is the way *git* thinks about its data [15]. Conceptually, most other systems store information as a list of file-based changes. Other systems like *Subversion*, *Perforce*, *Bazaar*, etc. think of the information they store as a set of files and the changes made to each file over time (see Figure 3.1).

Instead, *git* thinks of its data more like a series of snapshots of a miniature file-system (See Figure 3.2). With *git*, every time you commit or save the state of your project, *git* basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, *git* does not store the file again, just a link

[2] `https://insights.stackoverflow.com/survey/2018`

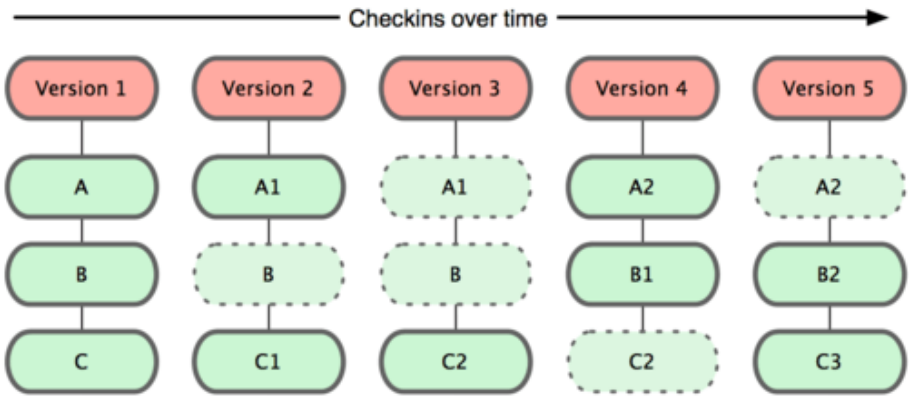


Figure 3.2: How *git* structures its information internally [17].

to the previous identical file it has already stored. All this information is stored in a key-value system as *git* objects, with a unique identity for each of them.

3.3 GitHub

GitHub is a *git* and web-based repository hosting service founded back in 2008. While *git* is a command line tool, GitHub provides a graphical interface, adding its own collaboration features such as a wikis and basic task management tools for every project.

Each user on GitHub has their own profile, showing its past work and contributions to other projects via *pull requests*, *forking* (create editable copies into someone's account) other repositories, etc. Project revisions can be discussed publicly via *issues* so many people can collaborate together to advance a project forward. GitHub is currently the largest host of source code in the world (see Figure 3.3 for a quantitative estimation of its scale evolution across time) [2].

3.4 GHTorrent

GHTorrent is a project aimed to create a scalable, queriable, off-line mirror of data offered through the GitHub *API* [4].

As they explain in their website[3], *GHTorrent* monitors the GitHub public event time line. For

[3] `http://ghtorrent.org/`

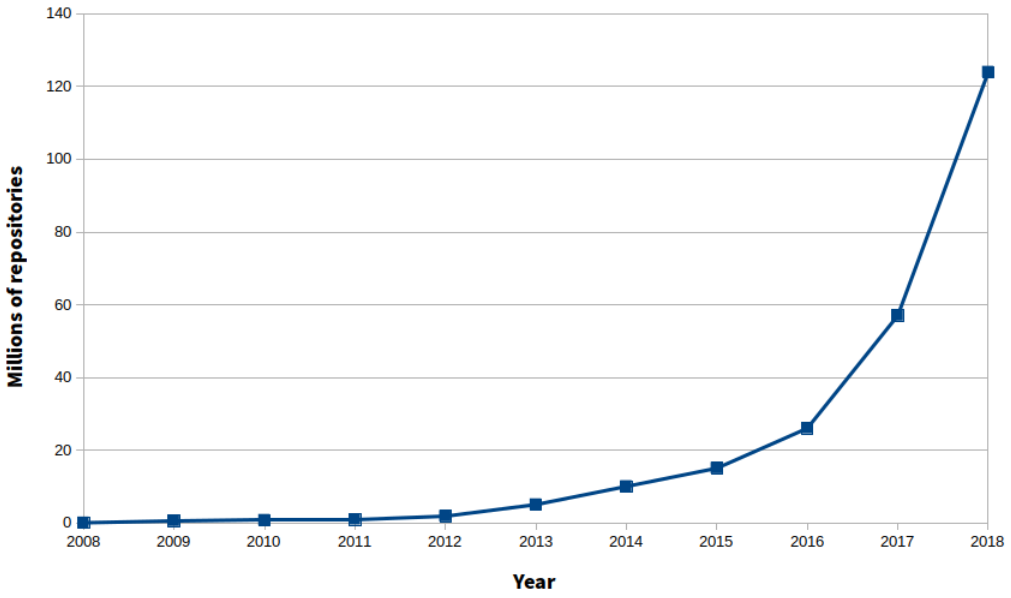


Figure 3.3: Estimation: Evolution of number of GitHub repositories across time

each event, it retrieves its contents and their dependencies, exhaustively. Then, it stores the raw responses to a database whose structure is represented in Figure 3.4. For each release, you can choose the *dump* (a raw copy of a database) you want to download: either a *MySQL* dump (the full database, using one file per table) or a *MongoDB* one (an incremental database).

3.5 GrimoireLab and Perceval

GrimoireLab[4] is a free, open source tool-set for software development analytics mainly developed by the Spanish company *Bitergia*[5]. It allows you to retrieve data from many kinds of systems with information related to software development, and produce analysis and visualizations with it.

We are focusing on a particular tool of *GrimoireLab* called **Perceval** [3]. This tool can retrieve data from more than 20 different kinds of data sources, from *git* repositories or GitHub projects, to issue trackers such as *Jira* or *Bugzilla*, including messaging systems such as *IRC*, *Slack* or mailing lists, or other types of systems such as *StackOverflow* or *Jenkins* in a regular and

[4] `http://grimoirelab.github.io/`
[5] `https://bitergia.com/`

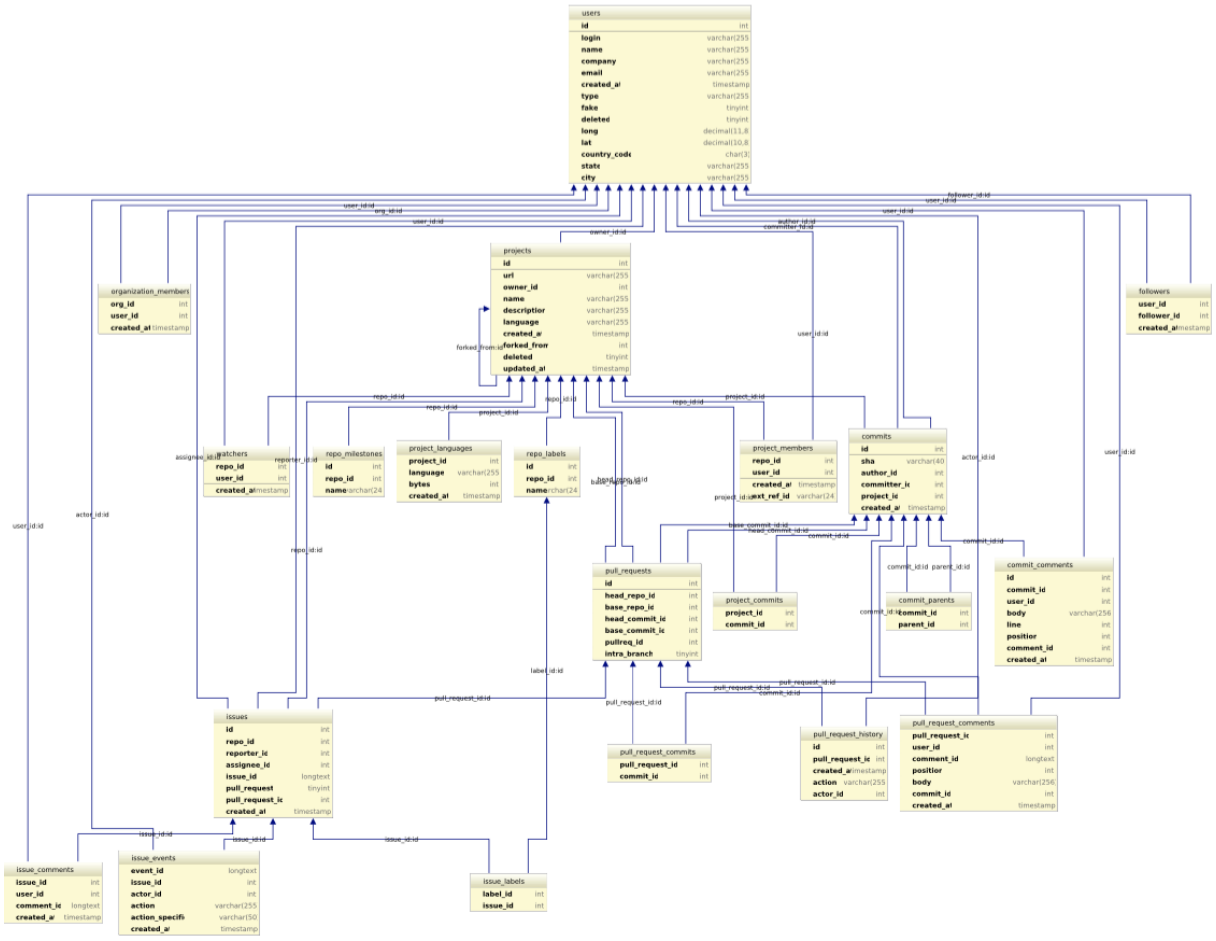


Figure 3.4: GHTorrent database relational schema

incremental way, allowing to produce uniform sets of information.

GrimoireLab is now part of *CHAOSS*[6], a project by *The Linux Foundation*.

3.6 MySQL

MySQL is an open-source relational database management system based on Structured Query Language (SQL), which is the most popular language for adding, accessing and managing content in a database. It is most noted for its quick processing, proven reliability, ease and flexibility of use.

MySQL works in multiple platforms, running as a server and allows multiple users (MySQL *clients*) to manage and create numerous databases.

It is a central component in the LAMP stack of open source web application software (LAMP stands for Linux, Apache, MySQL, and PHP) and it is adequate to be used in high-profile, large-scale systems and websites. To mention some examples, MySQL is used by Facebook, Twitter, Flickr and YouTube.

[6] `https://chaoss.community/`

Chapter 4

Design and implementation

The tool has a modular design, to ease its adaptability to future updates, changes on it dependencies or other needs which may appear. To fully understand the functioning of the tool, it is necessary to be familiar with main *git* objects (commits, branches and trees), as they are going to play a key role in this project. In Section A.1, a complete definition for these *git* concepts can be found. It is also important to mark that the research which motivated this tool influenced most of the decisions which were taken during the design phase.

The tool is divided into several linear sections with a set of *Python* scripts. We decided to code this tool in Python because it was the programming language I feel more comfortable with, along with its versatility due to the great amount of compatible packages and modules, followed with a great amount of documentation.

The set of scripts which compounds this tool[1] can be grouped into three main phases (see Figure 4.1). Before going into detail in each of these stages, let's introduce them briefly:

1. **Preliminary phase**

    - **Extract project list**: In this preliminary phase is where the list of GitHub repositories is retrieved. Using an off-line dump of the *GHTorrent* database, we get the *project list* file containing all GitHub repositories from one of its tables and use a filter to choose among projects by their fields, like a major programming language,

[1] See more details about the code of this tool in appendix B.

forked or not, etc.

2. **Data extraction**

   - **Extract file list**: This is the most critical process, because it where we obtain the main *branch* and the file list (for each repository in the list from last section) querying the GitHub *API*.

   - **Filter projects**: It consists on iterating over the extracted file lists, applying the corresponding patterns and heuristics. This produces a *list of positive results* file (repositories with at least one match).

3. **Data analysis**

   - **Analyze positive projects**: Its function is to execute *Perceval* with every project on the list of positive repositories, building a *projects database* which can be queried to extract information.

4.1 Preliminary phase

Given the scale of GitHub, the main problem we face is that there is not a direct way of obtaining the amount of repositories in the platform, neither getting a list of them using its *API*. Thus, it is very difficult to have quantitative and reliable data. For example, for building a graph like Figure 3.3, some of its numbers had to be obtained directly from articles written by the GitHub team[2] and some others by reading some GitHub-related papers containing quantitative data [5]. At this point it is where *GHTorrent* project plays a major role in this project, as they provide an off-line database dump with most of the data which the GitHub *API* offers at a certain date. This great collaborative effort is a huge advantage for researchers, otherwise it would be practically non-viable retrieving full-scale GitHub data in a systematic way. Nevertheless, it is important to mark that the *GHTorrent* database dump does not contain *Git trees* (i.e., file-related) information: if this information was included, a major part of this tool would not be necessary, as we could query and filter the data of our interest directly from the database provided by the *GHTorrent* project.

[2] `https://blog.github.com/2013-12-23-10-million-repositories/`

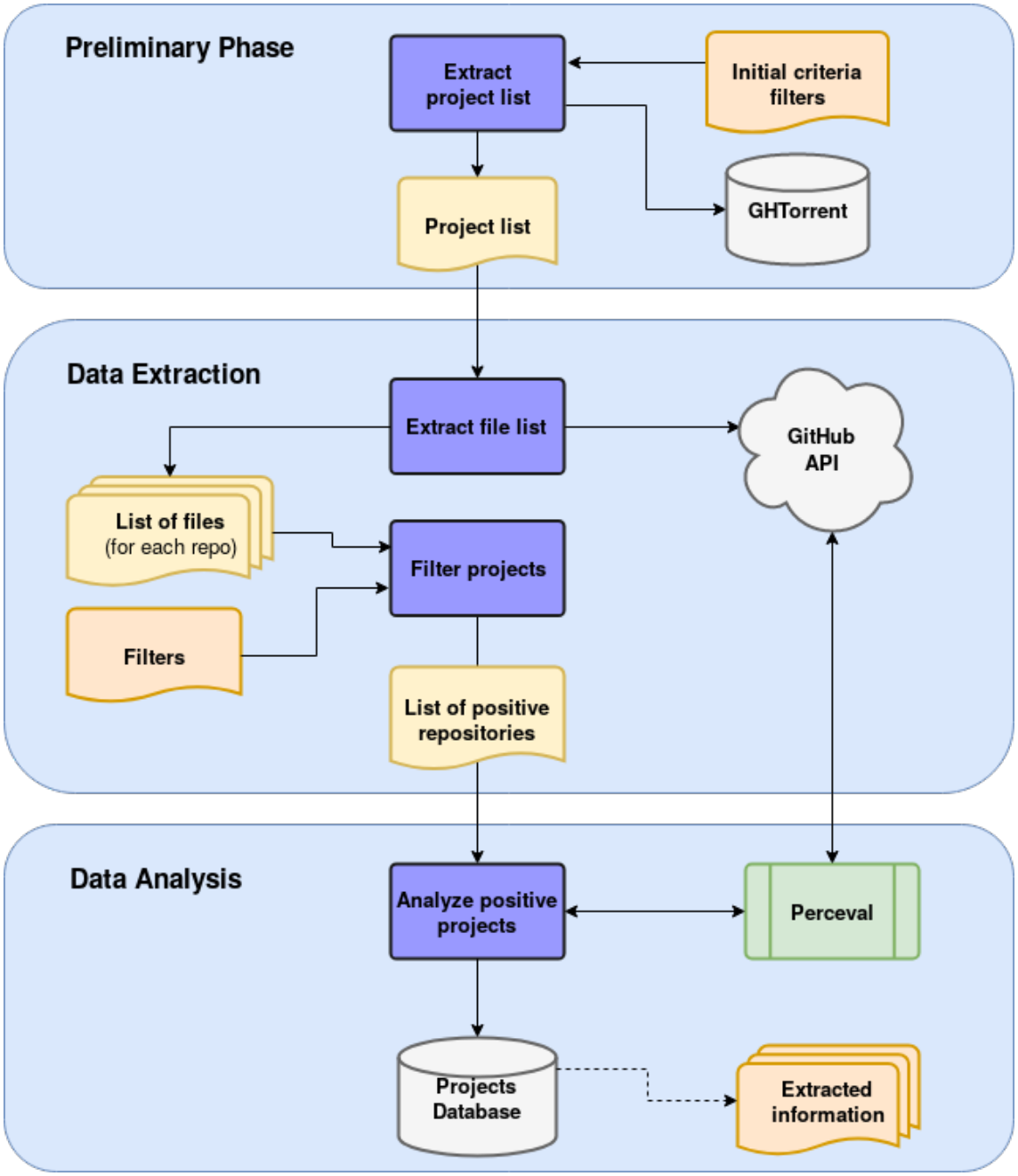


Figure 4.1: General architecture of the tool

Table 4.1: Adapted extract from `projects.csv` file, formatted as a table

id	owner	owner_id	name	descriptor	language	created_at	forked_from	deleted	updated_at
20	chrisjaure	80	git-lava	Branching metaphor for git	Shell	2012-02-27 02:45:44	-	0	2015-10-18 01:39:38
21	ES-DOC	82	django-cim-forms	-	Python	2012-03-23 14:20:28	-	1	0000-00-00 00:00:00
23	adammark	86	Markup.js	Powerful JavaScript templates	JavaScript	2011-08-23 00:30:04	-	0	2015-10-12 10:46:04
24	leoamigood	88	1stdibs_V2.1	Initial setup with mule-pull project	Java	2012-05-03 11:43:31	-	1	0000-00-00 00:00:00

If we have a look at how the *GHTorrent* database is structured (see Figure 3.4), we can observe that there is almost one table per GitHub object (*users*, *projects*, *pull requests*, *issues*, etc.). Looking closer on these table's structure, we realize the table we need is the *Projects* table (see Figure 4.2), as it contains crucial repository-wise information including every project's name and *URL*, but also other complementary information which could be used as a primary filter, like `language` (project's main language, code-wise) or `forked_from` (Empty if the repository is not a *fork* from another project), etc.

To obtain the actual data, the procedure to follow is as follows:

1. Download from the *GHTorrent* website a particular *MySQL dump*, usually the most recent one, which is provided as a compressed file in `tar.gz` format[3].

2. Uncompress the `tar.gz` file, which contains one *CSV* file per table in the database.

3. Get the file `projects.csv`, whose data corresponds to the *Projects* table (See an adapted[4] example of this file in Table 4.1).

4. Run the Python script `get-project-list.py` setting up the corresponding filters and parameters to obtain the main *project list* file, which will be the input for the next stage.

Note that the filtering we establish in this phase is going to influence the results of our analysis. Finally, we obtain a filtered *project list* file, which keeps the same format as `projects.csv` of the database dump.

[3]To give an estimation of the current file sizes, the last available dump (March 1st 2018) has a compressed size of 71,025 `MB`.

[4]The field *url* has been replaced by *owner* to make the table more readable.

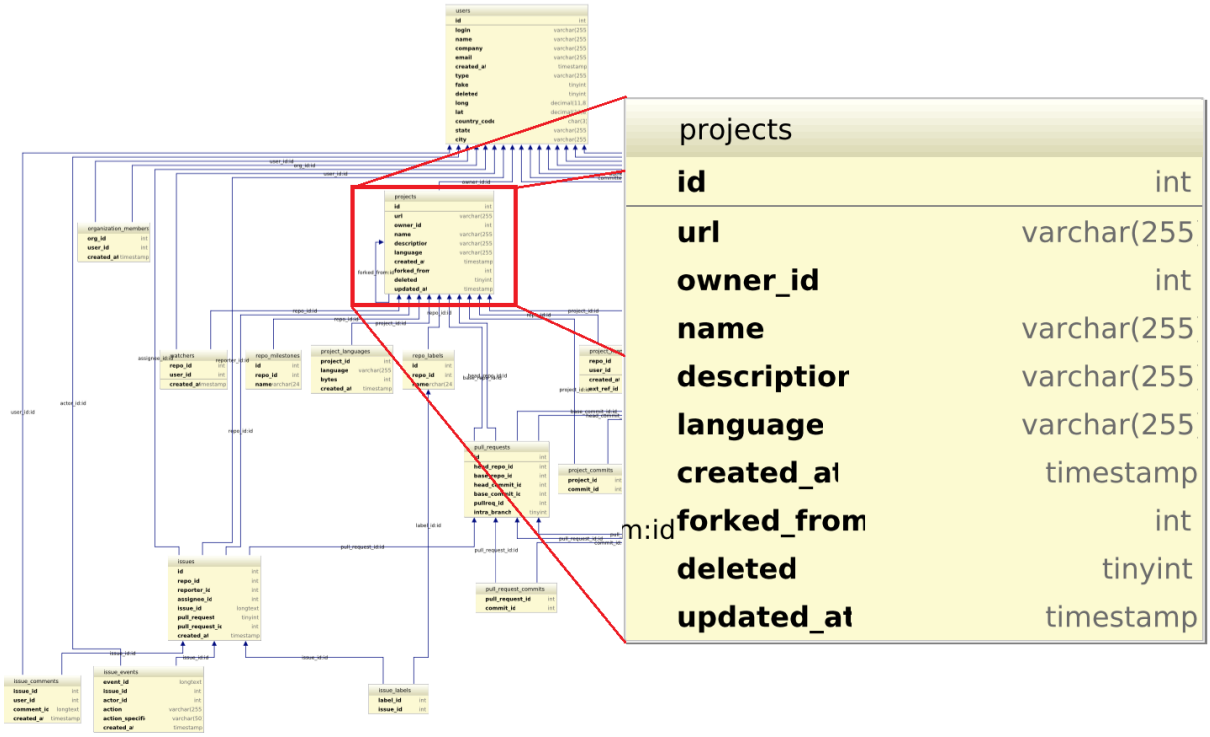


Figure 4.2: *Projects* table structure from GHTorrent database

4.2 Data extraction

Once the *project list* file is ready, we can proceed to extract the file list of the main *branch* of each repository. To achieve this, we have to query the GitHub *API* several times for each project. It is important to mark this, as the GitHub *API* has a limitation of 5,000 queries per hour for every authenticated account, which caused a huge impact in the research and the data retrieval process, as it is detailed in the case of study described in Section 5.1.

To make authenticated queries to the GitHub *API* we need an **API token**[5]. For instance, if I want to *fork* a repository into my GitHub account, I could do it either in two ways:

- Log-in into my account, visit the repository *URL* and press the **fork** button on GitHub's web interface.
- Send an authenticated *query*[6] to the GitHub *API* like this one[7]:

```
1 POST https://api.github.com/repos/:owner/:repo/forks?:token
```

[5] See the definition for API token in Subsection A.2.2

[6] From now on, every time I refer to a GitHub *API* request I will omit `https://api.github.com`.

[7] On the requests, the parameters are marked with two dots (:).

The script for this phase, `github-api.py` (See its diagram in Figure 4.4) takes into account this API limitation, ensuring that this limit is not exceeded. It employs three different directories in order to classify its outputs: **master**, **default** and **trees**. Then, it reads the *project list* file and, per line, executes the following actions:

- Check if the repository corresponding to that line has been downloaded before by looking at the existing files in the directories **master** or **default**.
- If not, try to obtain data from the *master branch*, whose output will be saved into the directory **master**. The *query* sent to the GitHub *API* to retrieve this data is:

```
GET /repos/:owner/:repo/branches/master?:token
```

  If we obtain a successful response, we will have a `JSON` file similar to the one in Figure 4.3.

- If the *master branch* is not found, we have to obtain the default *branch* for that repository and perform another *query* to retrieve the data from that *branch*, which will be saved into the directory **default**. To retrieve this information, we need to obtain meta-data from the repository first and then, keep the content of the variable `default_branch` to perform another *query* to obtain data from that *branch*:

```
GET /repos/:owner/:repo?:token
GET /repos/:owner/:repo/branches/:default_branch?:token
```

- The last step is to read the obtained `JSON` data and get the tree objects recursively using the ID (*SHA hash*) of the first one found in the response:

```
GET /repos/:owner/:repo/git/trees/:sha?recursive=1&:token
```

  This will be saved into the directory **trees**.

When this script finishes, we will have (assuming there were no errors) a `JSON` file per repository, with the file-list information for the *tree* objects in each repository.

However, the design of this script would not be complete without considering all the possible errors and particularities that may appear. Below, it is shown the most relevant setbacks and, if possible, an alternative solution for every one of them:

- **Private repository**. Some repositories in GitHub can be private if their owner hires a special plan on GitHub. The only solution is to perform the request with a token with granted access permissions to that repository, otherwise private repositories are ignored.
- **Truncated response**. Some repositories are too large so their tree and blob information is not completely sent within the response, but only a part of it and a Boolean field called `truncated` set to `True`. This was something that the GitHub *API* implemented while we were performing the data retrieval of our use case, so we had to adapt the tool for it. The solution is to *clone* (download) the repository, so later the next script can iterate locally over its files and folders.
- **Repository no longer exists**. There are repositories which existed at the time when that particular dump of the GHTorrent database was created, but they do not exist anymore.
- ***Charset*-related errors**. Either the name of the repositories, owners, files, etc. can be written using different types of characters (Japanese, for instance) or other unknown characters (using another encoding) which sometimes caused encoding-decoding errors.

4.3 Data filtering

In this section is where the data we have obtained from the previous stage is filtered according to our search. The approach that have been followed to apply filters is to define three types of heuristics, related to file extensions and keywords:

- **Level-1** file extensions, those which are a top priority in our search. They are immediately considered as positive results.
- **Level-2** file extensions, marked as relevant only if they meet one or more additional conditions.
- **Key-words**, specific words or groups of characters which a file-name with a *Level-2* extension has to contain to be considered as a positive result.

Let's imagine a basic example where we are interested in projects related with web development. Defining the necessary heuristics as in Table 4.2, we will keep any file whose extension

```
{
  "name": "master",
  "commit": {
    "sha": "f3d88db7d46...",
    "commit": {
      "author": {
        "name": "mghfdez",
        "email": "maf...@...",
        "date": "2015-01-11T21:05:32Z"
      },
      "committer": {
        "name": "mghfdez",
        "email": "maf...@...",
        "date": "2015-01-11T21:05:32Z"
      },
      "message": "Address PEP8 messages",
      "tree": {
        "sha": "88b3f83e3475...",
        "url": "https://api.github.com/repos/mafesan/ptavi-pfinal/git/trees/88b3f83e3475..."
      },
      "url": "https://api.github.com/repos/mafesan/ptavi-pfinal/git/commits/f3d88db7d46...",
      "comment_count": 0,
      "verification": {
        "verified": false,
        "reason": "unsigned",
        "signature": null,
        "payload": null
      }
    },
    "url": "https://api.github.com/repos/mafesan/ptavi-pfinal/commits/f3d88db7d46...",
    "html_url": "https://github.com/mafesan/ptavi-pfinal/commit/f3d88db7d46...",
    "comments_url": "https://api.github.com/repos/mafesan/ptavi-pfinal/commits/f3d88db7d46.../comments",
    "author": null,
    "committer": null,
    "parents": [
      {
        "sha": "22f7c6377bb3...",
        "url": "https://api.github.com/repos/mafesan/ptavi-pfinal/commits/22f7c6377bb3...",
        "html_url": "https://github.com/mafesan/ptavi-pfinal/commit/22f7c6377bb3..."
      }
    ]
  }
}
```


Figure 4.3: Example: `JSON` response to the query asking for master branch data

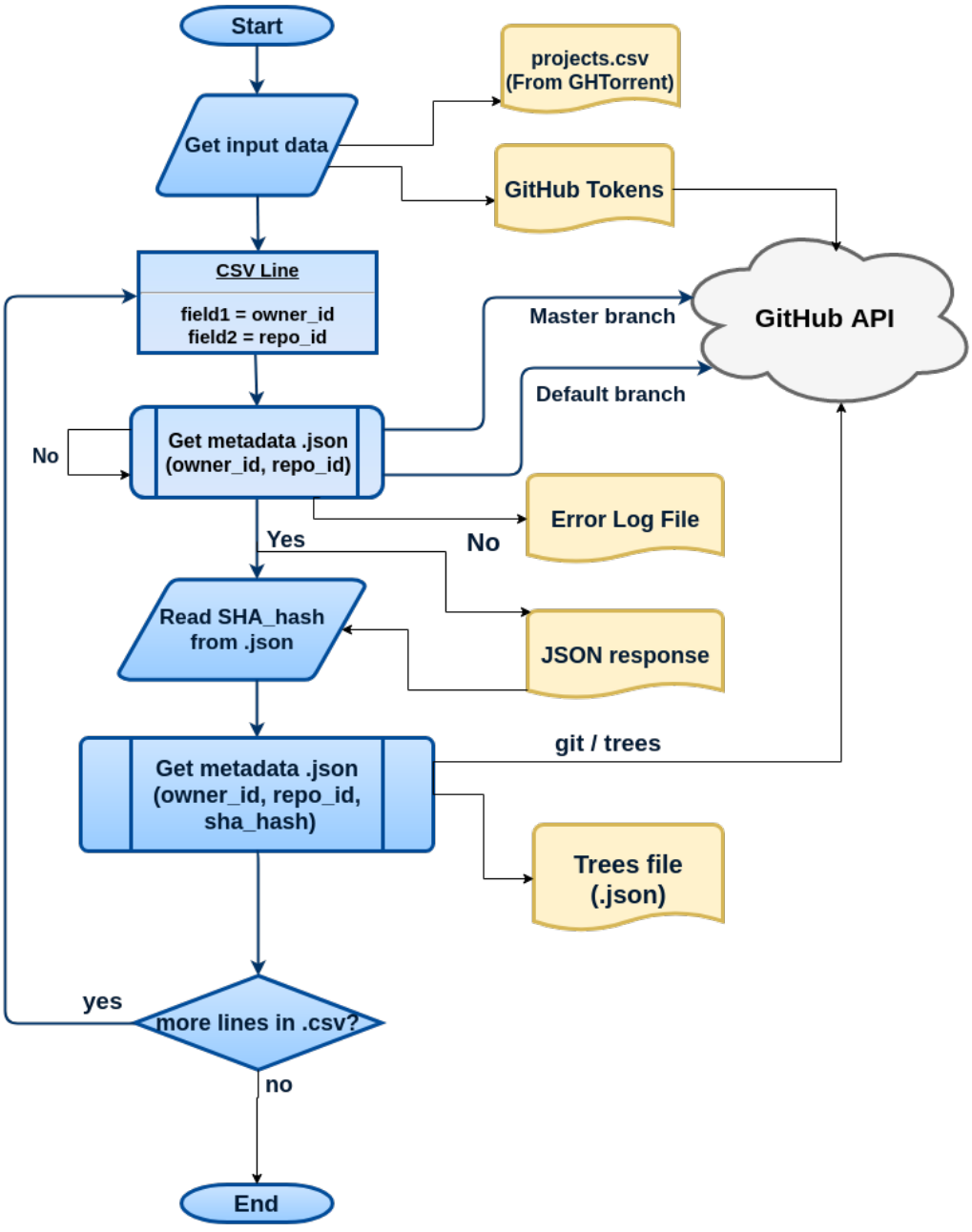


Figure 4.4: Flow-chart of `github-api.py` script

is on Level-1 list, and those files whose extension is on Level-2 list **AND** whose file-name contains at least one of the words (or group of characters) defined in the key-word list.

- `.html` is on **Level-1** list, so any file with `html` extension will be a positive result directly.
- `.py` is on **Level-2** list, so only `.py` files containing at least one key-word in their file-name (i.e. *urls*) will be considered as a positive result.

To complete this example, if we apply this filter to a repository whose file structure is like the one in Figure 4.5, we would obtain the set of positive results showed at Table 4.3.

Now, the next step is applying these filters to the `JSON` files which have been obtained before. Specifically, we are interested only in those files included into the **trees** directory. Each of those files contains the file structure of the last version of its repository, where the *tree* object contains under itself all the files (*blob* objects) and sub-directories (other *tree* objects). Every object, no matter if it is a *tree* or a *blob*, contains the following parameters:

- **path**: Absolute path of that file or folder inside the repository.
- **mode**: This number shows file mode information (file type, permissions, etc.) using UNIX notations.
- **type**: This value will be `blob` if it is a file, or `tree` if it is a folder.
- **sha**: Unique identifier for that *git* object.
- **size**: File size in *bytes*. Only `blob` objects contain this parameter.
- **url**: Link to this object in GitHub *API*.

There is a simple example of the content of one of these files in Figure 4.6.

The corresponding scripts for this phase are `github-tree.py` and `hits2urls.py` (See a diagram for both of them in Figure 4.7). `github-tree.py` iterates over all the `JSON` files in the **trees** directory, executing these simple instructions per file:

- Load `JSON` data into a *Python dictionary* (a key-value data structure).
- Obtain the parent `tree` object (field "`tree`" in our *dictionary*).
- Check the "`type`" field for every child object.

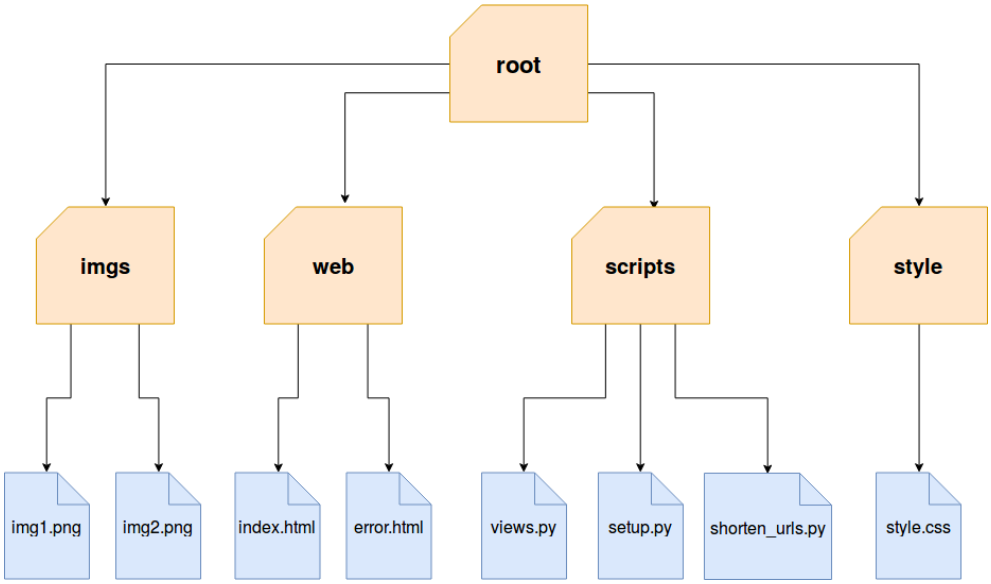


Figure 4.5: Example: File-structure of a project to apply filters.

Table 4.2: Example: definition of heuristics.

Type of Heuristic	**Pattern(s)**
Level-1	*.html*, *.css*, *.js*
Level-2	*.py*
Key-words	*urls*, *views*, *forms*, *models*

- If its type is "`blob`", apply the pre-defined patterns and heuristics.

- Provide the positive results printing every matching object's "`path`" and "`URL`".

`hits2urls.py` is a simple script which converts every `URL` from the last script (links to `blob` objects in the GitHub *API*), to the `URL` for the actual file which that object is representing. Here is an example of one `URL` which belongs to a `blob` object converted to the `URL` pointing to the actual file (see below lines 1 and 2, respectively):

```
1 https://api.github.com/repos/mafesan/ptavi-pfinal/git/blobs/3eb76189a21a...
2 https://raw.githubusercontent.com/mafesan/ptavi-pfinal/master/uaclient.py
```

```
{
  "sha": "88b3f83e3475...",
  "url": "https://api.github.com/repos/mafesan/ptavi-pfinal/git/trees/88b3f83e3475...",
  "tree": [
    {
      "path": "check-pfinal.py",
      "mode": "100644",
      "type": "blob",
      "sha": "683db02c4031...",
      "size": 6801,
      "url": "https://api.github.com/repos/mafesan/ptavi-pfinal/git/blobs/683db02c4031..."
    },
    {
      "path": "proxy_registrar.py",
      "mode": "100644",
      "type": "blob",
      "sha": "dd691148a13b...",
      "size": 20043,
      "url": "https://api.github.com/repos/mafesan/ptavi-pfinal/git/blobs/dd691148a13b..."
    },
    {
      "path": "ua1.xml",
      "mode": "100755",
      "type": "blob",
      "sha": "994061c5e45e...",
      "size": 265,
      "url": "https://api.github.com/repos/mafesan/ptavi-pfinal/git/blobs/994061c5e45e..."
    },
    {
      "path": "uaclient.py",
      "mode": "100755",
      "type": "blob",
      "sha": "3eb76189a21a...",
      "size": 9000,
      "url": "https://api.github.com/repos/mafesan/ptavi-pfinal/git/blobs/3eb76189a21a..."
    },
    {
      "path": "uaserver.py",
      "mode": "100644",
      "type": "blob",
      "sha": "c59b736d9117...",
      "size": 9982,
      "url": "https://api.github.com/repos/mafesan/ptavi-pfinal/git/blobs/c59b736d911717..."
    }
  ],
  "truncated": false
}
```


Figure 4.6: Example: JSON file containing information about *tree* objects

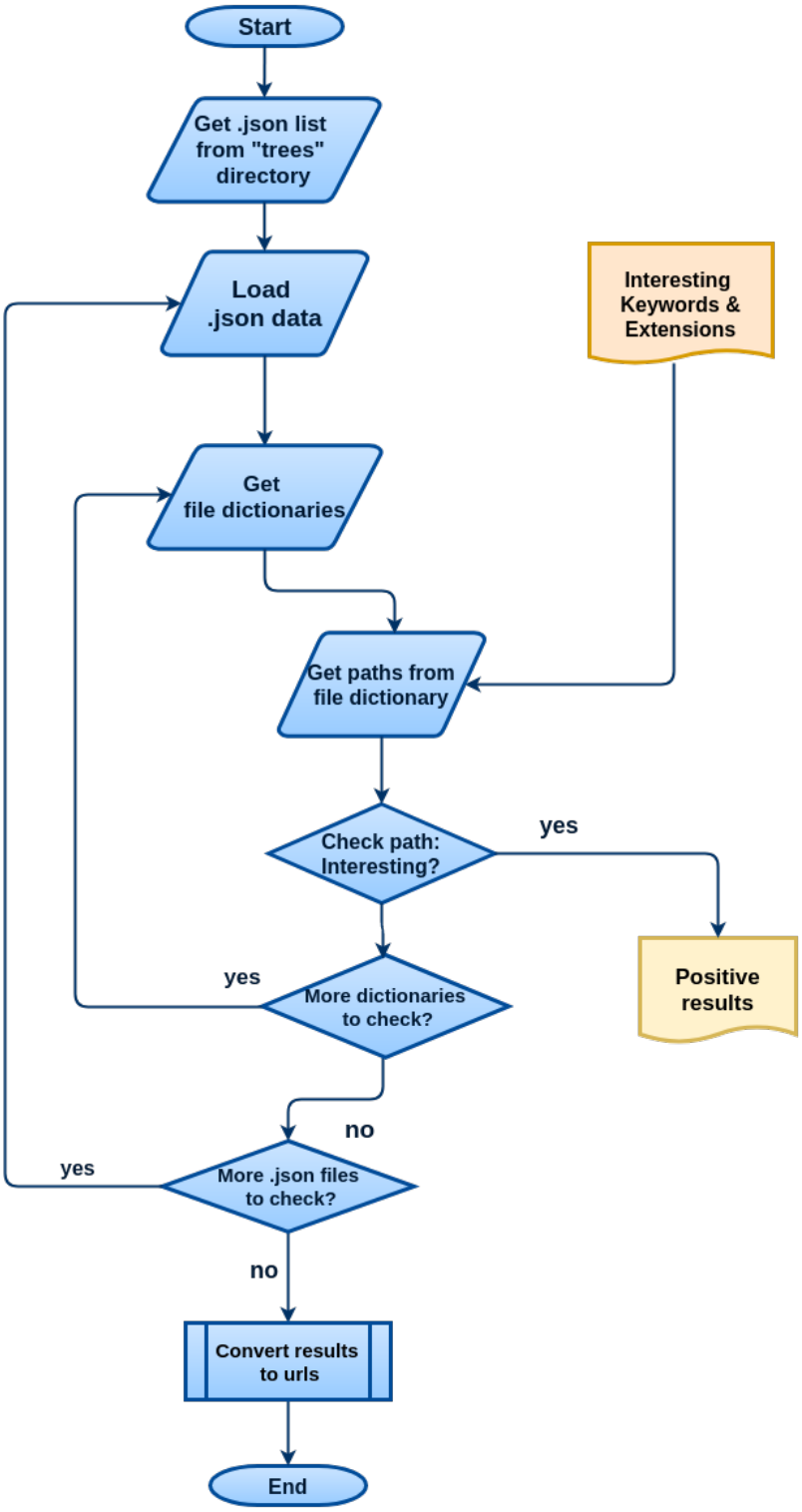


Figure 4.7: Flow-chart with the functioning of `github-tree.py` and `hits2urls.py` scripts.

Table 4.3: Positive results after applying heuristics filter.

Path	Type of positive
scripts/shorten_urls.py	Level-2 + contains key-word
scripts/views.py	Level-2 + contains key-word
style/template.css	Level-1
web/error.html	Level-1
web/index.html	Level-1

4.4 Data analysis

In this section, the positive results are analyzed to end up building a database to store all the extracted information so it can be queried to perform our desired analysis.

4.4.1 Extract extended repository information

To analyze the resulting repositories we take advantage of **Perceval**, a mature, powerful tool which is capable of retrieving data from more than 25 different data sources producing a uniform set of data which can be updated over time.

The data sources which Perceval supports are given by its *backends*. We use the *Git backend* to obtain Git-related information from those GitHub repositories with at least one positive result. Perceval clones repository by repository and parses every Git event log, producing a *JSON* file per repository containing data for every commit and its related Git objects. This tool is written in the Python language and it can be executed via command line or as a Python module, so it can be easily integrated with this set of scripts. In our tool, the script managing Perceval execution is called `perceval-handler.py`.

4.4.2 Building the database

The next step consists in building a database, which allows to store the extracted data in an optimized way to obtain elaborated results by using queries according to our case of study.

As we had used the GHTorrent database, which follows a MySQL database schema, we decided to use a MySQL database for this purpose too, as it is easier to build and maintain. We designed this database to contain five tables; four of them using data from the files which are obtained with **Perceval** and one table from the GHTorrent database, `USERS` containing information about GitHub user accounts. These tables contain the following information:

- `REPOS`: For a repository; its name, number of commits, URL, founder, etc.
- `PEOPLE`: Referring to the people who authored the commits, their name and email.
- `COMMITS`: For each commit object; its ID in Git, its ID on GitHub, which repository belongs to, ID of its author, etc.
- `INTERESTING_FILES`: For each file (positive result); its name, URL, which repository belongs to and which commits this file is included into.
- `USERS`: From GHTorrent, GitHub user account information as login, name, location, company, creation date, etc.

After a first version of this database, it was decided to add an additional table, *FILE_COMMITS*, which contains augmented information about each file commit, indicating the commit unique ID, the name of the committed file and the field file-type (type of the committed file based on a predefined classification, e.g., source code, documentation, etc.). These tables are represented in the complete schema of this database in Figure 4.8.

The dedicated script which converts the analyzed data into a database, `projects2sql.py`, produces one SQL script per table. Then, these scripts have to be imported into a MySQL database whose structure has to be specified before. Once the database has been built and filled with the data, we have to ensure to perform the correct queries to produce proper results for our analysis. For instance, here are some queries we can execute to perform an analysis:

- Name of positive repositories with more than 200 commits and whose last commit has been made after July 2, 2017:

```
1  SELECT name FROM repos
2  WHERE number_commits > 200
3  AND last_commit >= "2017-07-02";
```

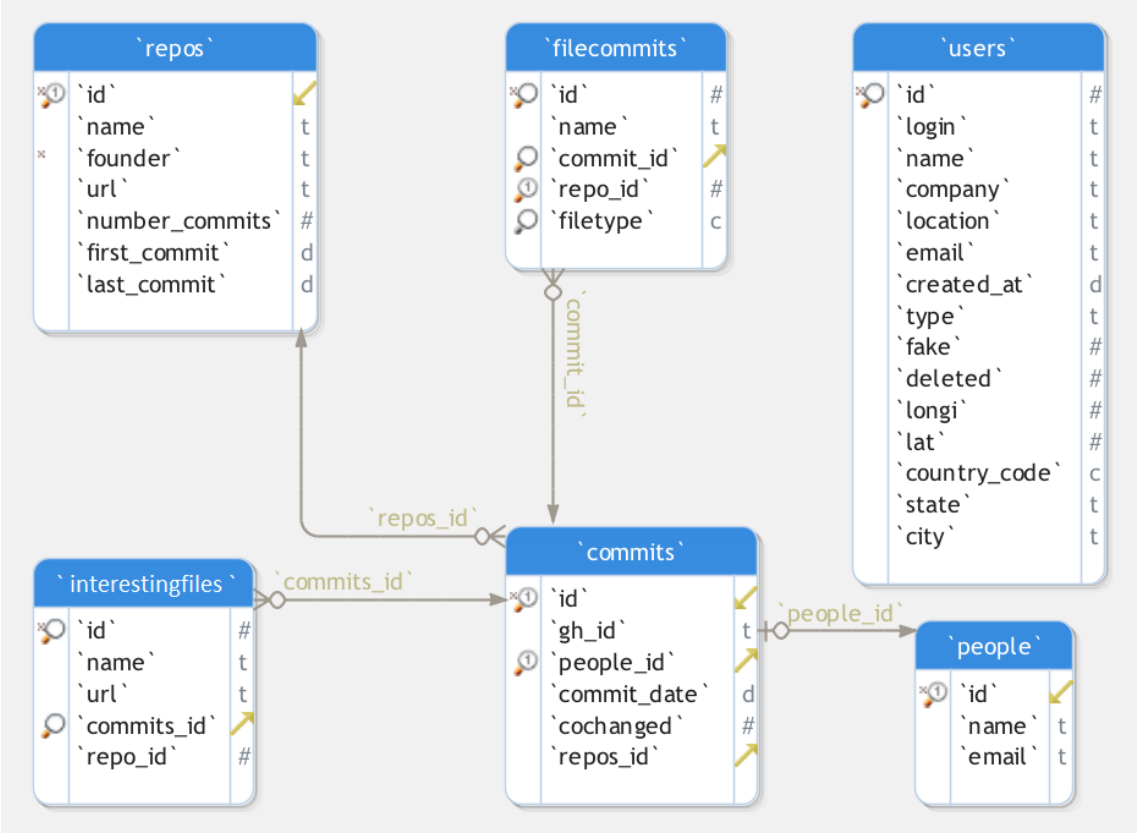


Figure 4.8: Database schema from the data-analysis phase.

- Number of repositories with at least one file with *.png* extension:

```
SELECT COUNT(DISTINCT repos.name) FROM interesting_files
LEFT JOIN repos ON interesting_files.repo_id = repos.id
WHERE interesting_files.name LIKE '\%.png';
```

Chapter 5

Results

As I mentioned in the Introduction (see Chapter 1), this project was born in a research environment. In this section, how the tool behaved in a real environment is described, with all the setbacks and limitations that appeared during the whole process. We will remark that the ongoing research influenced most of the decisions which were took during the design and implementation of the tool that we saw in the last section, as well as how to overcome the different drawbacks.

The results presented here led to several scientific papers published in international Software Engineering conferences which are highly ranked according to the CORE[1] portal:

- ICSE (International Conference on Software Engineering) - Rank A* (Source[2]).
- MODELS (International Conference on the Unified Modeling Language) - Rank A (Source[3]).
- MSR (International Conference on Mining Software Repositories) - Rank A (Source[4]).

The papers [6, 8, 13] are available in Appendix C.

[1] http://www.core.edu.au/conference-portal
[2] http://portal.core.edu.au/conf-ranks/1209/
[3] http://portal.core.edu.au/conf-ranks/1244/
[4] http://portal.core.edu.au/conf-ranks/711/

5.1 Case of study: UML models in GitHub projects

The method described in Chapter 4 was applied for the first time in the study published in the paper "The quest for Open Source projects that use UML: Mining GitHub" [6] (The full paper is available in Appendix C.1). This research has been a collaborative work between Chalmers and Rey Juan Carlos University in the context explained in Section 1.1. The main goal of this study was 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. In addition to the main research questions mentioned in Chapter 2, some of the specific RQs 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?
- RQ3: When in the project are new UML models introduced?
- RQ4: What is the time span of "active" UML creation and modification?

It is important to know if there is some peculiarity 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 Figure 5.1). This entails an additional problem as an external validation of the data is required in an intermediate step between the data filtering phase and the analysis one, which will be detailed later.

The first two phases of the tool, described in Sections 4.1 and 4.2 respectively are, by default, independent from the case of study. However, these phases are not a trivial task because of the amount of data that has to be processed with the difficulties and limitations associated with it.

Regarding to the preliminary phase, the GHTorrent dataset used in this study is from February 1st 2016. At that time, there were a total of 26 million repositories hosted in GitHub (approx.). As the main research that motivated this study was interested in identifying 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 analyzed.

As already noted, the greatest challenge was to execute the tool against almost 13 million GitHub repositories, as the 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:

- 3 queries for each project = 1,666 projects/hour (max).
- 12,847,555 projects / 1,666 projects per hour = 7,712 hours = 322 days

We made an estimation where 322 days would be needed with the best scenario, so we opted for paralleling the process using many different GitHub accounts.

To make this possible, we asked for help to some students and professors so they can temporarily borrow their GitHub account token and we obtained 21 account tokens. Finally, it took almost 90 days to complete this task as we were receiving these credentials while we were already collecting data[5]. 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 after the data extraction stage we got 126 GB of compressed JSON files, which have to be opened and processed.

To continue with the data extraction phase, the next step is to apply the filter to the file-list information we had obtained. In this case, we are interested in looking for files whose extension matches with all possible extensions which an UML model can be found, as is it showed in Table 5.1.

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? As the tool has a modular design, it was easy to add an additional phase to verify the data which was obtained after the filtering. This verification process was solved by Chalmers team, using heuristics for the XML-alike files and image processing and machine learning techniques for image files [7].

UML models can be included either in image or text files (see Figure 5.1), therefore our main

[5]I take this opportunity to thank every person who had the kindness to contribute to this project with their GitHub accounts.

filter using file extensions and character patters in file-names is not enough to completely identify UML files. That is why it was necessary the inclusion of an additional phase where all the positive outcomes after applying the first filter (now marked as **potential files**) need to be validated to determine whether they contain a UML model or not. This external validation stage developed by the *Chalmers* team includes machine-learning and digital image processing techniques which also required manual verification steps, which were very time-consuming.

This handicap led to the decision of shorten the data-set to analyze $\sim$10% of the total amount of GitHub repositories which resulted in 1,240,000 repositories, in order to offer accurately verified results. From those repositories, we obtained 100,702 potential files which matched after applying the heuristics filter in Table 5.1. These potential files are the input of the validation stage whose process and results are summarized on Figure 5.2: A total of 21,316 UML files were validated included in 3,295 different GitHub repositories, which is the answer to RQ1.

Next, in the data-analysis phase these projects were processed with Perceval and a database was built using that information. Once the database was created it was obtained more elaborated data, for instance, the distribution of positive repositories according to how many UML models contained each of them (see Table 5.2) but also the answers to the rest of the RQs:

- RQ2: Are there GitHub projects in which the UML models are also updated?
  - Only 26% of this first data-set updated their UML files at least once.
- RQ3: When in the project are new UML models introduced?
  - UML models are introduced in all active phases of a project with a tendency towards the early phases.
- RQ4: What is the time span of "active" UML creation and modification?
  - Few of the studied projects are active with UML during their whole lifetime. In general, the projects work very shortly in UML, usually at the beginning.

Then, the validation process described before was extended to the whole set of potential results, the UML search was finished. These complete results were presented in the paper "An extensive dataset of UML models in GitHub" (The full paper is available in Appendix C.2), which resulted

in the identification of 93,596 (see Table 5.3) publicly available UML models in GitHub from over 24,125 repositories. Though this amount of repositories only represents 0.19239% of the total amount of the initial set, as I anticipated in the Introduction (Chapter 1), the identified models constitute a dataset that is two orders of magnitude larger than the rest of UML data-sets at the time of this research.

Applying the data-analysis phase to the complete UML models data-set we ended up with a database with deeper information about these results, which broaden the possibilities to enrich different aspects of this research. This derived in the paper "Practices and Perceptions of UML Use in Open Source Projects" (the full paper is available in Appendix C.3), whose goal of this later research was "to shed some light into the motivation and benefits of the use of modeling and its use within project teams". To do so, the applied method was to perform a survey among open source developers, focusing on projects that use UML as a representative for software modeling.

The constructed database played a pivotal role for this survey, as it was necessary to define which projects and developers met the survey conditions executing the right queries to the database. There were two main lines to obtain the survey target:

1. **Filtering short-time projects out**: For this paper, researchers aimed at projects that are interesting from an industry perspective. Then, we focus on projects that are **not** short-term and that do not jut consist of a singe contributor. In this context, the accurate definitions of "short-time project" are:

   i. Active time (time between the first and the latest commits) less then 6 months, OR

   ii. less than 2 contributors, OR

   iii. less than 10 commits.

   After classifying and filtering short-time projects out, 4,650 UML projects (out of 24,125) met the requirements.

2. **Identify developer role in a repository**: It was important to consider the role that the different contributors play within the OSS projects with UML models to ask them the appropriate questions, considering a combination of roles in the following two dimensions:

```
<?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>
```

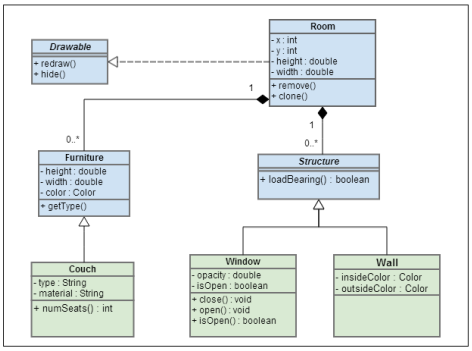


Figure 5.1: Examples of how UML models can be found, text-based (left) or image type (right).

Table 5.1: Example: definition of heuristics for the case of study 1.

Type of Heuristic	**Pattern(s)**
Level-1	*.uml*, *.xmi*, *.uxf* and *.xdr*
Level-2	*.xml*, *.bmp*, *.jpg*, *.jpeg*, *.gif*, *.png* and *.svg*
Key-words	*xmi*, *uml*, *diagram*, *architecture* and *design*

i. Founder (F) vs. non-founder (NF)

ii. Non-UML Contributor (NUC) vs. first UML Contributor (1UC) & UML Contributor or updater (non-1st contributor) (UC)

Therefore, each interview participant had to fulfill one of the following 6 roles: F-1UC, F-NUC, F-UC, NF-1UC, NF-NUC, NF-UC.

Furthermore, researchers considered the possibility of having duplicate identities in our database, as one contributor can use different emails, user-name (for example, changing the user-name or email during the project time). The email and the full-name were the primary keys for this check, which resulted in 99,319 distinct contributors out of 129,276 original ones.

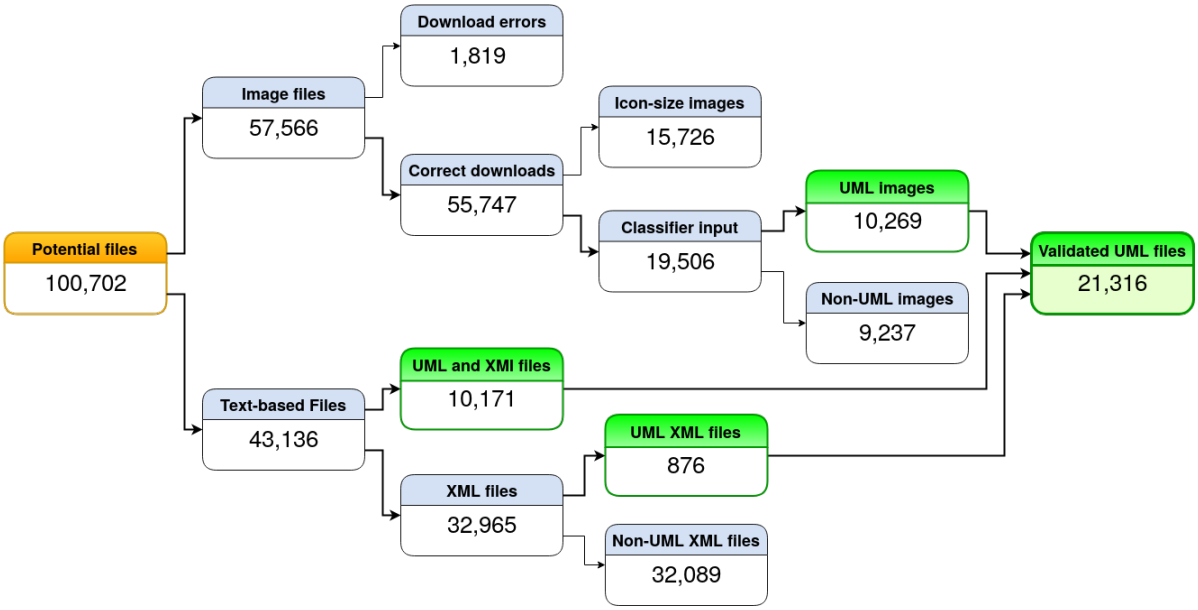


Figure 5.2: Diagram: Distribution from potential files with UML to validated UML files.

Table 5.2: Validated positive repositories (partial) sorted by their number of UML models.

Number of UML models	Number of repositories
1	1,947
From 2 to 9	1,169
From 10 to 99	158
100+	4

Table 5.3: Complete results of validated positive UML models sorted by their type.

Type of model	Number of UML models
Text-based (.uml, .xmi)	35,774
Image-based	57,822

Table 5.4: Example: Definition of heuristics for the case of study 2.

Type of Heuristic	**Pattern(s)**
Level-1	*.uml*, *.xmi*, *.uxf*, *.xdr*, *.eap*, *.aird*, *.argo*, *.asta*, *.dfClass*, *dfUseCase*, *.ecore*, *.ecorediag*, *.umlcd*, *.mdj*, *.plantuml*, *.simp*, *.txvcls*, *.umlx*, *.ump*, *.uxf*, *.zargo*, *.zuml*, *.zvpl*, *.dia*, *.modelproj*, *.classdiagram*, *.sequencediagram*, *.activitydiagram*, *.usecasediagram*, *.componentdiagram*, *.layerdiagram*, *.cd*, *.di* and *.umldi*
Level-2	*“”*, *.xml*, *.bmp*, *.jpg*, *.jpeg*, *.gif*, *.png*, *.svg*, *.txt*, *.doc*, *.docx*, *.ppt*, *.pptx* and *.pdf*
Key-words	*xmi*, *uml*, *diagram*, *architecture*, *design*, *sad*, *sdd*, *arch* and *hier*

5.2 Case of study: Software Architecture documents & extended UML models in GitHub projects

After the derived studies from the first dataset, the second case of study came out when the same research team was interested in extending the search looking for models with other less-restrictive extensions, but also looking for software architecture documents, as sometimes the models from a project are defined inside a document (i.e., a PDF file).

Accomplishing this new case of study was substantially easier and faster, as all the JSON files containing file-related information from all GitHub projects had already been downloaded. Then, with the preliminary and data extraction phases completed, the procedure to follow was to execute the steps from the data-filtering stage using the patterns and heuristics from Table 5.4. This data-filtering phase resulted in a list of 6,373,748 potential files belonging to 573,854 different GitHub repositories. Since these potential results have not been validated yet, no further phases of the tool were executed.

Chapter 6

Conclusions

During the whole implementation and the later set-up process of the tool with the different use-cases, several handicaps, limitations and setbacks were found which affected how the objectives described in Chapter 2 were reached. Nonetheless, looking back at the tool and how it performed with the different use cases, along with the obtained results, it is safe to say the purpose of this tool was achieved, providing answers to the proposed research questions.

6.1 Achieved objectives

Regarding OB1[1], we were able to extract the whole set of projects hosted in GitHub at a certain date and time thank to the *GHTorrent* project. This is the first threat following this approach: during the time where the MySQL dump of *GHTorrent* from GitHub data is created and the data extraction phase is executed, the source of information is constantly changing, so we may encounter some projects that do not exist anymore, some projects become private and of course, there are new projects that are being created. These limitations are hard to avoid but also it is safer to extract the information from an static, reliable and uniform source.

About OB2[2], this was the most arduous objective to accomplish, as setting up and watching over the different parallel instances (up to 21 instances running at the same time) to speed-up

[1]Extract the whole set of projects hosted in GitHub.

[2]Collect data from all public projects about their file structure in a scalable, automated way.

the completion of the whole data-retrieval process required a huge amount of time and effort. The GitHub API limit of 5,000 requests per hour & account was a huge drawback we tried to avoid contacting to GitHub asking them for a special account without limitation in order to perform the study, but the solution they gave us was to create more accounts and using them all to perform more requests per hour (and so we did). Furthermore, GitHub API changed during this data-retrieval phase including paginated results. That forced us to adapt this tool and re-analyze some of the data that have been already downloaded.

These drawbacks could be avoided in future versions of this tool if *GHTorrent* included git-trees information. It would save a lot of time and resources, together with having more reliability due to the staticness this data would have.

Looking at OB3[3], the positive results can be identified properly. However, due to the different particularities of the data we want to analyze, we may want to add more filtering layers to the obtained potential results. Technically, the most difficult issue was to handle huge JSON files (some of them sized GBs) and cover the diverse Charset-related errors.

OB4[4] was completely covered using *Perceval*, with the limitations of encountering positive repositories which had been deleted or set as private, reducing the final data-set.

Finally, OB5[5] was also completely accomplished. These two last objectives share some technical difficulties regarding to storage space and performance: Perceval needed to clone the repositories, which means it had to download and uncompress data; and some of the generated SQL files with the database information (like the one for the `commits` table) contained millions of entries, which aside from their size, hindered the importing process.

6.2 Knowledge application

During my degree I have learned about important concepts and tools during the different courses, but also about how to face new challenges where to implement the acquired knowledge. In this project I have applied the learning outcomes from the following courses:

[3] Establish a procedure to filter this extracted data using a determined kind of heuristics and patterns.

[4] Analyze positive GitHub repositories to obtain enhanced Git data from them.

[5] Store the enhanced data in a database whose structure allows to query information of interest.

1. **“Informática I”** (Fundamentals of Programming): This course was my first programming-related course, where I learned the basics of programming” (basic data and flow-control structures).

2. **“Informática II”** (Telecommunication Systems Programming): In this course I learned advance programming techniques along with more complex data structures, including for the first time communication protocols as UDP and TCP connections elaborating both client-server and peer-to-peer applications.

3. **“Arquitectura de Internet”** and **“Sistemas Telemáticos para Medios Audiovisuales”** (Computer Networks I and II): Thank to these courses I learned about both basic and complex communication protocols and how the Internet works, from TCP/IP protocol to HTTP connections, routing algorithms and protocols, etc.

4. **“Protocolos de Transmisión de Audio y Video en Internet”** (Audio/video Transmission Protocols): In this course I learned how to program in Python language and the basic concepts for object-oriented programming, in addition to real-time communication protocols and basic Git concepts.

5. **“Laboratorio de Tecnologías Audiovisuales en la Web”** (Web-based Technologies): Despite the fact this course is not directly related, I learned basic concepts about databases and I fostered my programming skills in Python building my first Django application.

6.3 Learning outcomes

These are some of the learning outcomes I have reached thank to this project:

1. Fostering of my programming skills, especially in Python. Though I was familiar with this programming language in particular, I have learned about general, complex concepts of programming as parallelization and concurrency but also specific advanced Python concepts and structures: sets (unordered collections of non-duplicated values), list comprehensions (a way to to create lists containing an expression followed by a for clause, then zero or more for or if clauses) and other Python *idioms*.

2. Learning new technologies, specially about databases and SQL language which complements my academic background. I have also learn how to interact with APIs for the first time.

3. A great overview about research. Since I started to work as a researcher assistant, I was surprised about how little I know about research and its procedures about scientific papers, conferences, etc. With this project I was able to learn from the very beginning the stages of the elaboration of a study to its final stages including the process for the paper to be published.

4. A valuable perspective about team work and communication, having the possibility to collaborate with an international team from a different university.

5. System administration experience. Taking care of the technical tasks about this project forced me to learn how to execute and watch over all the stages of this tool in different GNU/Linux distributions, mostly using a remote server.

6.4 Future work

The tool and its execution process can be improved in several aspects. Regarding to the preliminary and data-extraction stages, these are some of the features that could be implemented:

- A method to keep updated the initial set of GitHub projects from new dumps from *GHTorrent*.
- If *GHTorrent* dumps format change, adapt the necessary scripts to ensure compatibility.
- A method to keep updated the JSON files containing the file-list information for each repository, and another to unify the format of that data in those repositories whose response was truncated.
- Multi-thread execution support to have different parallel instances of `github-api.py` script.
- Improve monitoring and exception management.

In the data-filtering and the data-analysis phases, the proposed improvements are:

- Add more filter types or improve the existing ones.
- Improve the method to analyze the positive repositories using Perceval. There is a new *GrimoireLab* tool called **Arthur**[6], which is a distributed job queue platform that schedules and executes Perceval using threads, error management and more. This new tool could be integrated in the data-analysis stage to set the repositories to be analyzed by Perceval in a more efficient way.
- Optimize how the analyzed data is imported into a new database even exploring other different database types.

6.5 Personal assessment

I had the great luck to be in the right place at the right moment when I started this project. Working on it has brought me a lot of great experiences and outcomes, which are extensive to my period working as a researcher assistant in the GSyC/LibreSoft group where I met fantastic people.

I feel fulfilled after having participated in such an important research and see its resulting scientific papers presented and published in important conferences. I have obtained more knowledge but also self-confidence to face large-scale projects, to make presentations in front of an audience and also to foster my English level. Furthermore, I had the opportunity of traveling to Gothenburg (Sweden) in June 2016 to meet the team from Chalmers University we were collaborating with. Afterwards, this cooperation provided me the chance to enroll myself in an Erasmus+ traineeship in that university during Summer 2017.

[6] `https://github.com/chaoss/grimoirelab-kingarthur`

Bibliography

[1] M. R. Chaudron, W. Heijstek, and A. Nugroho. How effective is uml modeling? *Software & Systems Modeling*, 11(4):571–580, 2012.

[2] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.

[3] S. Dueñas, V. Cosentino, G. Robles, and J. M. Gonzalez-Barahona. Perceval: Software project data at your will. In *Proceedings of the 40th Intl Conference on Software Engineering*, ICSE'18, 2018.

[4] G. 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.

[5] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman. Lean ghtorrent: Github data on demand. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 384–387, New York, NY, USA, 2014. ACM.

[6] R. Hebig, T. H. Quang, M. R. V. Chaudron, G. Robles, and M. A. 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.

[7] T. Ho-Quang, M. R. V. Chaudron, I. Samúelsson, J. Hjaltason, B. Karasneh, and H. 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.

[8] T. Ho-Quang, R. Hebig, G. Robles, M. R. Chaudron, and M. A. Fernandez. Practices and perceptions of uml use in open source projects. In *Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), 2017 IEEE/ACM 39th International Conference on*, pages 203–212. IEEE, 2017.

[9] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of the 11th working conference on mining software repositories*, pages 92–101. ACM, 2014.

[10] J. Moreno-León, G. Robles, and M. Román-González. Examining the relationship between socialization and improved software development skills in the scratch code learning environment. *J. UCS*, 22(12):1533–1557, 2016.

[11] G. Robles, J. M. González-Barahona, D. Izquierdo-Cortazar, and I. 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.

[12] G. Robles, J. M. Gonzalez-Barahona, and J. J. 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.

[13] G. Robles, T. Ho-Quang, R. Hebig, M. R. Chaudron, and M. A. Fernandez. An extensive dataset of uml models in github. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 519–522. IEEE Press, 2017.

[14] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified modeling language reference manual, the*. Pearson Higher Education, 2004.

[15] N. B. Ruparelia. The history of version control. *ACM SIGSOFT Software Engineering Notes*, 35(1):5–9, 2010.

[16] R. Somasundaram. *Git: Version control for everyone*. Packt Publishing Ltd, 2013.

[17] B. Straub and S. Chacon. *Pro Git*. Apress, second edition, 2014.

[18] M. Tepper. The rise of social software. *NetWorker*, 7(3):18–23, 2003.

[19] G. Van Rossum et al. Python programming language. In *USENIX Annual Technical Conference*, volume 41, page 36, 2007.

[20] B. Vasilescu. Software developers are humans, too! In *Proceedings of the Companion Publication of the 17th ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW Companion '14, pages 97–100, New York, NY, USA, 2014. ACM.

Appendix A

Definitions

A.1 Git objects definitions

A.1.1 Commit

A **commit** is a *git* object which contains a record of the changes made to the repository since last modified version of itself (last commit object). In the example in Figure 3.2, each **commit** would be each different *Version* (1, 2, 3...), meaning that there is a new version of the repository with every **commit**.

A.1.2 Tree

A *git* **tree** represents a directory and its structure (including file-names) in the repository, containing other possible sub-directories as other child **tree** objects. However, these objects do not contain any information about the file contents: that is stored in **blob** objects. In Figure A.1 there is an example of how **commits**, **trees** and **blobs** are connected with each other.

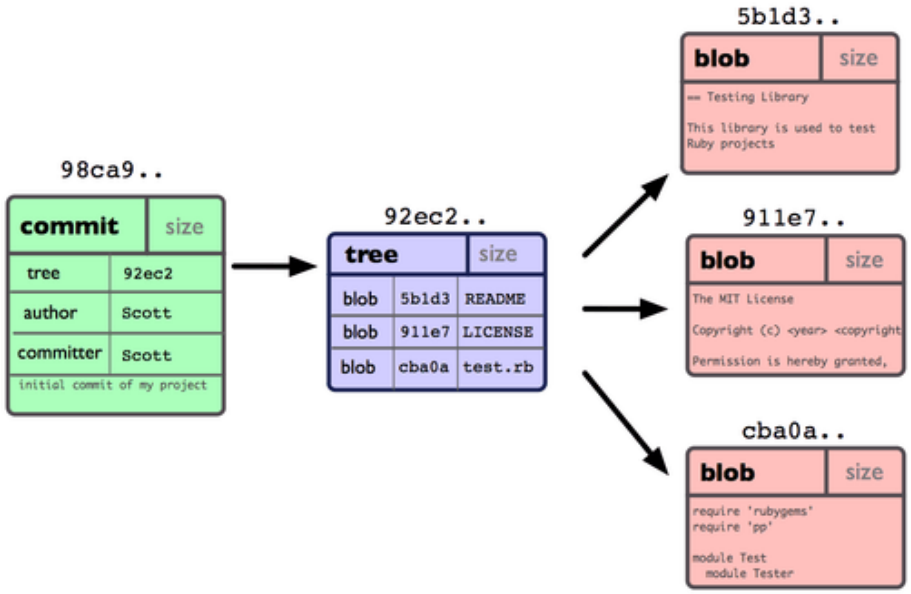


Figure A.1: Interaction among **commits**, **trees** and **blobs** (unique IDs above) [17].

A.1.3 Branch

A *git* **branch** is a pointer to a certain *commit* object. All new commits created from this pointer will diverge from the previous commit-history of the repository in two distinct, independent paths. Branching is a very common and useful practice, as it allows to isolate different versions of the same files and directories (but also different) in the same repository, and later, those branches can be merged into any other branch (See in figure A.2[1]) a simplified example about the structure of a repository with several branches).

A.2 API

An **API** (*Application Programming Interface*) is a set of subroutine definitions, protocols, and tools for building application software. In general terms, it is a set of clearly defined methods of communication between various software components.

[1]Source: `https://www.atlassian.com/git/tutorials/using-branches`

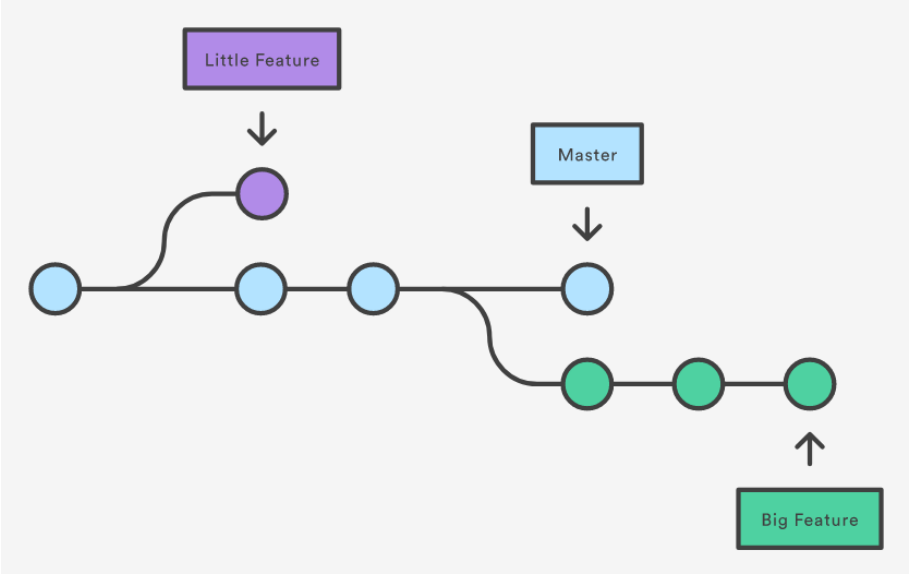


Figure A.2: Abstraction of *git* **branches** structure in a repository.

A.2.1 GitHub API

GitHub *API* allows to access GitHub data, since own GitHub types like *pull-requests*, *issues* or *forks* to *git*-related data, such as *commits*, *branches* or *trees* using *HTTP* requests and returning information using *JSON* (*JavaScript Object Notation*) format.

A.2.2 API token

A **token** is a unique identifier of an application requesting access to a service. In the case of GitHub API, they are long, alphanumeric strings of characters generated within a GitHub account, so every executed action with that **token** is done on behalf of its owner's account.

A.3 Essential freedoms of Free/Libre Software

A program is free software if the program's users have the four essential freedoms:

- The freedom to run the program as you wish, for any purpose (freedom 0).
- The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1).

- The freedom to redistribute copies (freedom 2).
- The freedom to distribute copies of your modified versions to others (freedom 3).

A program is free software if it gives users adequately all of these freedoms. Otherwise, it is *nonfree* or *propietary*.

Appendix B

Code of the tool

The tool presented in this thesis consists on a set of Python scripts and a SQL script to set up the database structure for the "Data analysis" phase. The last version of the code of this tool, together with a short execution manual is publicly available on the following GitHub repository: `https://github.com/mafesan/2018-tfg-code`.

To give an estimation about the magnitude of this tool, in table B.1 it is presented a software metric generated using "SLOCCount"[1], a set of tools for counting physical source lines of code (SLOC).

Table B.1: Source Lines of Code (SLOC) for the scripts compounding the tool

Phase	**Script name**	**SLOC**
Preliminary	get-project-list.py	91
Data extraction	github-api.py	126
Data filtering	github-tree.py	111
Data filtering	hits2urls.py	107
Data analysis	perceval-handler.py	109
Data analysis	perceval2sql.py	228
Data analysis	ghtorrent-users2sql.py	101
Total		873

[1] `https://www.dwheeler.com/sloccount/`

Appendix C

Published papers

C.1 The Quest for OS Projects that use UML: Mining GitHub

This paper was published in the MODELS conference (Saint-Malo, France), in October 2016. The complete reference is following:

Hebig, R., Quang, T. H., Chaudron, M. R., Robles, G., & Fernandez, M. A. (2016, October). The quest for open source projects that use UML: mining GitHub. In Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (pp. 173-183). ACM.

The Quest for Open Source Projects that use UML

Mining GitHub

Regina Hebig, Truong Ho Quang, Michel R.V. Chaudron
Chalmers | Göteborg University
{hebig,truongh,michel.chaudron}@cse.gu.se

Gregorio Robles, Miguel Angel Fernandez
GSyC/LibreSoft
Universidad Rey Juan Carlos, Madrid, Spain
grex@gsyc.urjc.es, mafesan.nsn@gmail.com

ABSTRACT

Context: While industrial use of UML was studied intensely, little is known about UML use in Free/Open Source Software (FOSS) projects. Goal: We aim at systematically mining GitHub projects to answer the question when models, if used, are created and updated throughout the whole project's life-span. Method: We present a semi-automated approach to collect UML stored in images, .xmi, and .uml files and scanned ten percent of all GitHub projects (1.24 million). Our focus was on number and role of contributors that created/updated models and the time span during which this happened. Results: We identified and studied 21 316 UML diagrams within 3 295 projects. Conclusion: Creating/updating of UML happens most often during a very short phase at the project start. For 12% of the models duplicates were found, which are in average spread across 1.88 projects. Finally, we contribute a list of GitHub projects that include UML files.

Keywords

UML, open source, free software, GitHub, mining software repositories

1. INTRODUCTION

The Unified modeling language (UML) provides the facility for software engineers to specify, construct, visualize and document the artifacts of a software-intensive system and to facilitate communication of ideas [2]. For commercial software development, the use of UML has been introduced and commonly accepted to be a prescribed part of a company-wide software development process.

When it comes to Free/Open Source Software (FOSS) development, characterized by dynamism and distributed workplaces, code remains the key development artifact [1]. Little is known about the use of UML in open source. Researchers in the area of modeling in software engineering have performed some efforts to collect examples of models and of projects that use modelling. However the results are often limited [19]. For example, the Repository for Model Driven Development (ReMoDD)[5] is an initiative driven by an international consortium of leading researchers in the field of modeling. Nevertheless its content is growing at a low rate: after 9 years (summer 2016) it contains around 81 models. Industrial projects are very reluctant to share models because they believe these reflect key intellectual property and/or insight into their state of IT-affairs.

Due to the so far limited success in identifying open source projects with UML, many researchers (including the authors themselves at the start of this study) are rather pessimistic finding much use of UML in open source projects. Furthermore, since most open source platforms, such as GitHub, do not provide facilities for model versioning, such as tools for model merging, we were even more pessimistic about finding examples of UML models that were updated over time.

The lack of available data is the reason why so far no answers could be given to several basic questions on the amount of UML files in open source projects that are static or updated, the time span during which models are created or updated during the open source project, or the question which of the project's contributors do create models. Thus it seems that UML is not frequently present in FOSS projects. However, there is no exact quantification of its presence.

GitHub hosts around 10 million of non-forked repositories, which makes it a good starting point to obtain an estimation of the use of UML in FOSS projects. GitHub's web search is limited for this type of endeavor as it targets mainly source code searches by developers. While there are many other ways to access GitHub data (GHTorrent or the GitHub API) obtaining data on UML usage is not trivial (as we will show).

In this paper we present our efforts to mine GitHub in order to gain a list of open source projects that include UML models. Due to the required manual steps, it is not yet feasible to investigate all 12 million GitHub projects. Instead we focus on a random sample of 10% of all GitHub projects (1.24 million of the 12 million repositories). It turned out that for achieving this goal we required to join forces and expertise from different fields. The first challenge is the identification of non-forked repositories in GitHub with the help of the GHTorrent [6] in order to retrieve candidates for files that might include UML diagrams. Since these many of these diagrams are stored in formats that can also include other information than models, e.g. images or XML based files, it is further necessary to perform an automated recognition of those files that actually are UML. Therefore, it is required to perform two different checks, one for XML based

MODELS '16, October 02-07, 2016, Saint-Malo, France

 ISBN 978-1-4503-4321-3/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2976767.2976778

formats and one for images, which is a state of the research technology that just became available in 2014 [8]. Finally, with the retrieved list of UML models, the git repositories of these projects were accessed in order to retrieve information about the repositories and further information about commit and update histories of these models. As a result we gain out of over 1 240 000 repositories a first list of 3 295 projects containing UML models.

The contributions of this paper are: **1.** A first list of 3 295 GitHub repositories including altogether including 21 316 models. This list can be used by other researchers in future to find case studies and experimental data, e.g. for developing model versioning technologies or for studying how design decisions in models transfer to the code. **2.** Based on this data we give for the first time answers descriptive questions about the number of models that are subject to updates, the number of model duplicates that can be found, and the point in a projects life time where models are created and updated. **3.** Furthermore, this research provides the basis to ask when UML models are introduced and updated. Surely the approach has still limitations, for example we will not be able to identify how often the models are read. However, we believe that these first descriptive results are just a starting point. They enable us and other researchers to formulated and address more advanced questions about UML usage and its impacts on a project in future work.

The remainder of our paper is structured as follows. In Section 2, we formulate a number of research questions. Section 3 shows our review on relevant works. We describe our study approach in detail in Section 4. Our findings are presented and discussed in Section 5 and Section 6, respectively, including the threats to validity. We conclude our paper in Section 8.

2. RESEARCH QUESTIONS

The data set that we are assessing in this work would allow for a multitude of analysis, e.g. for assessing the distribution of different model types more precisely than it has been done in related work so far. However, answering all questions at once is not possible due to space limitations, but also due to limitation of time. Therefore, we decided to focus in this paper on a set of descriptive questions that had not been addressed in related work so far and that provide a necessary starting point and frame for future analysis:

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?*

These first two questions are interesting for two reasons. First, their answer represents a description of the state of practice that was simply not available so far. Second, projects with updates are ideal candidates for future investigations on model usage. For example, they might be used to evaluate facilities for model versioning.

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

Is it at the beginning of the project or later? What span of the project life time is covered by the phase where UML models are actively created or modified? Again the descriptive character of this questions is important. Only with the answer, we will be capable to formulate more precise questions on the model usage in future work. For example, whether these results are homogeneous amongst open source projects or not, will imply directions for future investigations. In long term/ future work this might lead to investigations what form of model usage is most efficient and so on.

RQ4: *What is the time span of "active" UML creation and modification?*

With this question we want to know how long is the time span during which models are in active use during a project? A limitation of our methodology is that we cannot investigate how often and when models are read. However, we can have a look at the time span of active UML creation and modification, i.e., the time between the first introduction of a UML file and the last introduction or update of UML files within a project.

RQ5: *Are UML files originals?* Special model versioning techniques such as model merging are not explicitly supported by GitHub. Therefore, we are interested in the question how many of the found models are duplicates of other models.

Despite the big interest in these questions, it was until now not possible to answer them. The reason is that simply no systematic knowledge exists about UML in open source projects. Furthermore, even if projects are known, it requires advanced mining of the repository in order to get related information about changes and contributors.

3. RELATED RESEARCH

This paper builds on previous research done in two research communities: the software modelling- and the mining software repositories communities.

3.1 Use of UML in FOSS

Studies on the usage of UML are frequently done amongst in industry (mostly through surveys) [16, 20]. However, only few studies focus on freely available models, such as can be found in open source projects. Reggio et al. [16] investigated which UML diagrams are used based on diverse available resources, such as online books, university courses, tutorials, or modeling tools. While this work was done mainly manually, Karasneh et al. [11] use a crawling approach to automatically fill an online repository[1] with so far more than 700 model images- Both works focus on the models only and do not take their project context into account. Further, they do not distinguish between models that stem from actual software development projects and models that are created for other reasons, e.g. teaching. An index of existing model repositories can be found online [19][2]. However, in addition to their small size, these repositories seldom include other artifacts than the models, making it impossible to study the models in the environment of actual projects.

Further, there are some works addressing small numbers of case studies of modeling in open source projects. Yatani et al. [23] studied the models usage in Ubuntu development by interviewing 9 developers. They found that models are forward designs that are rarely updated. Osman et al. [15] investigated 10 case studies of open source projects from Google-code and SourceForge that use UML. They focused on identifying the ratio of classes in the diagrams compared to classes in the code. They find only seldom cases where

[1]http://models-db.com/

[2]Index of model repositories http://www2.compute.dtu.dk/~hsto/fmi/models.html

models are updated.

Finally, there are three works that actually approach a quantitative investigation of models in open source projects. Chung et al. [3] questioned 230 contributors from 40 open source projects for their use of sketches and found that participants tend to not update these sketches. A study that focuses on software architecture documentation in open source projects was performed by Ding et al. [4] They manually studied 2 000 projects from SourceForge, Google code, GitHub, and Tigris. Amongst those projects that used such documentations they identified 19 projects that actually use UML.

The work that is probably closest to our study is the one of Langer et al. [12] They searched for files conforming to the enterprise architect file format (which is a format that can be used to store UML files) within Google code and GitHub. They identified 121 models. They further assessed the model lifespan (between introduction and last update) to be in average 1 247 days. However, studying a single file format is a rather limited view on UML. Furthermore, the project perspective is not considered and they rather put a focus on the used UML concepts.

3.2 Mining

Mining software repositories has mainly focused on aspects related directly to (programming) source code. However, projects may include non-source-code sources such as images, translation, documentation or user interface files, that can be usually identified by their extension [18]. By doing so, research has shed some light on the variation and specialization of workload that exist in FOSS communities [21].

The study of specific file formats that are non-source code can be found as well in the research literature: McIntosh et al. [13], [14] have investigated the build system for its evolution and effort, or the analysis of infrastructure as code that has become mainstream in the last years [9].

4. METHODOLOGY

In this section, we describe our study approach. The overall process is shown in Figure 1.

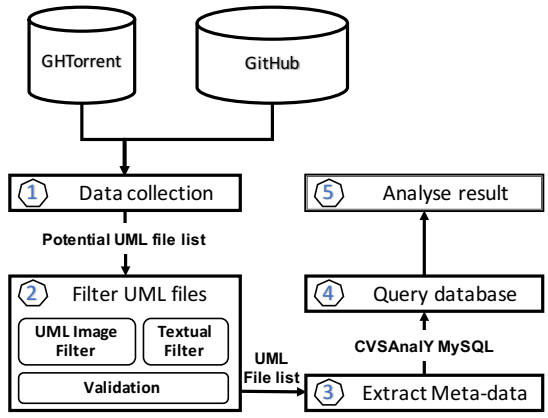


Figure 1: Overall process

First, we obtained a list of 10% of the GitHub repositories from GHTorrent [6] that are not forks. This resulted in a list of files of 1 240 000 repositories, those that had a branch that could be downloaded. From this list, potential UML files were collected using several heuristic filters based on the creation and storage nature of UML files (Step 1). Section 4.1 and Section 4.2 describe our approach and used filters in detail.

An automated process was built to examine the existence of UML notation in the obtained files (Step 2). A manual validation step is taken in order to consolidate the classification result. We describe the classification method in Section 4.3.

We have then obtained the meta-data from those repositories where a UML file has been identified by means of using the CVSAnalY tool [17] (step 3). Section 4.4 discusses tool's settings and the meta-data structure.

In step 4, we queried the metadata (taken in Step 3) with respect to our research questions. We answer the research questions by analyzing the result (Step 5). Note that during the data analysis further files got lost for diverse reasons (see discussion section 6). Thus, we were finally able to analyze a set of 21 316 UML model files.

A replication package of our analysis is available online [7].

4.1 Occurrence of UML

To understand how we searched for files containing UML, it is important to understand how these files are created and stored. Figure 2 illustrates the different sources of UML files (at the bottom in green). UML models might be created by manual drawing (sketching). Possibilities to create models directly with a computer are the usage of tools that have drawing functionality, such as Inkscape, or dedicated modeling tools, such as Modelio or Argo UML. Some of the modeling tools even provide the possibility to generate UML models, e.g. based on source code. This differences in tool support lead to a wide variety of ways in which UML models are represented by files. The different possibilities are illustrated in blue at the top of Figure 2: Firstly, manual sketches are sometimes digitized with the help of scanners or digital cameras and thus lead to image files of diverse formats. Secondly, tools with drawing capabilities can either store the UML models as images, such as .jpeg and .png or .bmp, or may have tool specific formats, e.g. "pptx". Thirdly, dedicated modeling tools work with tool specific file formats, e.g. the Enterprise Architect tool stores files with a ".eap" extension. Also some tools work with 'standard' formats for storing and exchanging UML: ".uml" and ".xmi". Yet, modeling tools with specific formats often allow to export and import these standard formats and allow to export the models as images. As a consequence, when searching for UML many different file types need to be considered.

4.2 Data Collection

For all repositories from GHTorrent [6] that are not marked as forks, we used the GitHub API: i) To obtain file list for `master` branch; ii) If no master branch found, ask for `default` branch; iii) To obtain the file list from `default` branch. With up to three GitHub calls (i, ii and iii) for each repository, given the GitHub API limitation of 5 000 requests/hour, it took over two weeks to retrieve the complete file list once the machinery was set up.

As explained in section 4.1, different file formats need to be taken into account. However, as not every image file is

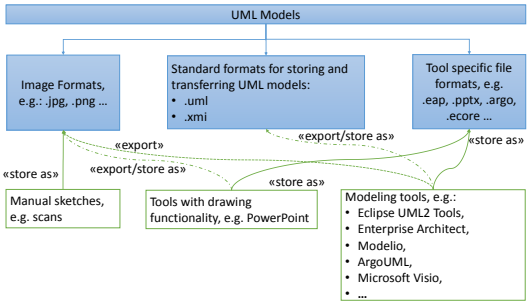


Figure 2: There is a large variety of tools for creating and formats for storing UML models

UML, also not every xmi file or files with the endings of tool specific format extensions are UML. Therefore, the filtering process does not only consist of the collection of files with a specific extension, but also of a check whether the collected files are really UML files. It makes no sense to collect files in the first step, for which we have no automated support for the second step.

Image files as well as standard formats are more common and are created by most modeling tools. For such common tools, developing an approaches to identify UML has a good cost-benefit ratio. The applied methods are explained below in section 4.3. However, for tool specific formats this ratio can be very low. Therefore, we searched only files of those formats where we could exclude two cases:

- The format is used within the tool exclusively for UML models.
- The file extension of the format is not used by other tools. For example the extension of Enterprise architect files (“.eap”) is also used for Adobe Photoshop exposure files.

To identify these formats we used as a starting point the list of UML modeling tools collected on WikiPedia[3], which we as experts consider as one of the most complete lists available. We checked whether the file formats used by these tools do not fulfill the two obstacles mentioned above.

Thus, we search for following file types:

- Images: Common filenames for UML files (such as ”xmi”, ”uml”, ”diagram”, ”architecture”, ”design”) that have following extensions (”xml”, ”bmp”, ”jpg”, ”jpeg”, ”gif”, ”png”, ”svg”)
- Standard formats: [”uml”, ”xmi”]

Hence we do not consider document formats such as word (.doc(x)), .pdf and powerpoint (.ppt(x)). The main reason is that currently technology is not yet capable of extracting UML models out of such general documents.

4.3 UML filters

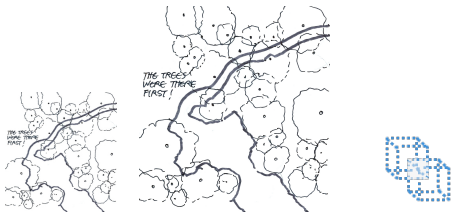
At this stage, the files obtained from Step 1 were checked if they really contain UML notation.

[3]List of modeling tools https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools, Last visited 9th December 2015

4.3.1 Identify UML images

Firstly, all images were automatically downloaded. Files that could not downloaded or unreadable were eliminated (Result: Successfully downloaded files downloads: 55 747; errors: 1 819). In addition, observations on downloaded images showed a remarkable number of icons and duplicate images. While it’s mostly impossible to find reasonable UML content in icon-size images, including duplicate images in candidate set could definitively cause redundancies to classification phase. Therefore, we eliminated icon-size images. Duplicate images were proceeded as: i) Duplicate images were automatically detected; ii) Representative images were added to classification candidate list; iii) After classification phase, duplicate images of an image will be marked as the same label as the image.

In particular, 15 726 images that have icon-dimension-size no bigger than 128 x 128 were excluded. Subfigures 3a, 3b and 3c show examples of such images.



(a) Dup 1a (b) Dup 1b (c) icon

Figure 3: Example of duplicates and icon-size images

In order to detect duplicate images, we created a simple detection tool by using an open source .NET library ”Similar images finder” [4]. Given two images, the tool calculates differences between their RGB projections to say how similar they are. In our case, we chose a similarity threshold at 95% since it gave the best detection rate through a number of tests on a subset of our images. Downloading of images took 27 hours.

The final image set of 19 506 images were classified as UML or non-UML images by using a classifier from our prior research [8]. The classifier was trained by a set of 1 300 images (650 UML-CD images and 650 non-UML-CD images). The Random Forest algorithm was chosen since it performed the best in term of minimizing the amount of false-positive rate (expecting below 4%). The automated classification too 26.5 hours. In order to eliminate false-positive and false-negative cases, we manually checked the whole image set. It took 6 working days of effort of an UML expert to complete the checking. This manual check allowed us to prove our classification method and to consolidate classification results. It turned out that the automated analysis had a 98.6% precision and 86.6% recall. The false positives and negatives could be identified due the the manual check.

Gradually, we manually picked up UML in other types (i.e., Sequence Diagram - SD, Component Diagram - CPD, Deployment Diagram - DLD, State Machines - SM and Use-case - UC). UML files that are sketches (SKE) were counted, too. The list of images was marked with a number of labels: ”UNREAD”, ”SVG”, ”SMALL”, ”DUP”, ”CD”, ”SD”, ”CPD”, ”DLD”, ”SM”, ”UC” and ”SKE”.

[4]https://similarimagesfinder.codeplex.com/

4.3.2 Identify UML files among .xmi and .uml files

Both .xmi and .uml files are specific XML formats. The later ones can include uml models, only and we found 10 171 of them. XMI is a standard format that should enable exchange of models between different tools. In theory it should be simple to identify whether an XML file in general contains a UML model: the schema reference in the XML file defines the content's format.

We performed the analysis in 3 steps:

1. In practice the schema reference are often generated in different forms by tools. For example, we found following three schema references to the UML: "org.omg/UML", "omg.org/spec/UML", and "http://schema.omg.org/spec/UML". Therefore we first of all searched with a simple search function for the string "UML" and "MOF" (the meta meta model of the UML language) in a random subset of the models. This way we could come up with a list of 7 strings representing UML schema references.
2. In a second step we automatically downloaded the identified xmi files and parsed them for the schema references. We could identify 876 files with UML schema references.
3. In a last step we wanted to double check that the existence of such a schema reference is sufficient to assume that the file includes UML. Therefore, we took a sample of four open source projects containing together 53 (between 1 and 33 respectively) links to xmi files. In addition to the check for schema references, we went manual through the content of the 53 files to assess whether and what kind of models they include. A comparison of the results with the data from the step above confirmed that the existence of an UML/MOF schema is a reliable indicator for rating a file as UML: of the 53 xmi files, 30 had been rated by both approaches as UML, while the other 23 were rated as non-UML.

Finally we run a duplicate detection on .xmi and .uml files by comparing hash values of the file contents.

4.4 Metadata Extraction and Querying

We downloaded all repositories where at least one (real) UML file was identified and extracted its metadata with the help of CVSAnalY [17]. 100 repositories from the initial list could not be retrieved, due to various reasons, e.g. changes from public to private repositories.

In average, around 30 000 projects per day were downloaded for each GitHub account. Taking these results a time span of 14 months ((12 847 555 projects / 30000) / 30) would be required for the analysis, when using one single GitHub account. As this would have made this study in feasible, we parallelized the retrieval of the JSON files through many GitHub accounts, which were donated during this process. This reduced the time span to approximately one month. While the download is an automated process, but the parallelization is not. It took around 1 h 30' each day to run and check each set of repositories, using up to 21 GitHub accounts. Altogether this process took 6 weeks.

After this process, we had 21 316 of the identified UML files from 3 295 repositories and the corresponding metadata in a SQL database. A new SQL table was added to the ones provided by CVSAnalY with just the UML files for easy and efficient querying. A set of Python scripts were used too query the database and aggregate the data required to answer the RQs. This final step took 14 days.

5. RESULTS

This section presents the results of our investigation. In this research an ample amount of data have been used, usually handled by scripts developed by the authors. Detailed information of the former and the code of the latter can be obtained in the replication package[5].

5.1 RQ1: UML in GitHub projects

We downloaded 1 240 000 non-forked GitHub repositories obtained from GHTorrent. After filtering the data for potential UML files based on type, we retrieved a list of 100 702 links. Of those, 21 316 were classified as UML.

The further extraction of model related data, turned out to be an additional filter, since details could not be extracted for all files. The reason for this is due to the fact that our retrieval procedure takes so much time that context changes. So, for instance, in the time that goes from the retrieval of information of the files the are included in a project (July/August 2015) to the time where the git repositories where downloaded (November/December 2015), some of them were renamed, deleted or made private.

In consequence, 21 316 files could be retrieved for the following analysis (as summarized in Table 1). These files belong to 3 295 GitHub projects. Of these 1 947 include a single UML file, only and 1 169 projects include between 2 and 9 UML files. Furthermore, we identified 158 projects with 10 to 99 UML files and 4 projects with more than 100 UML files. In the following analysis, the later 21 projects are taken separately, when statistics per model are shown. The reason is that they show very different characteristics and would, with their large number of models[6], strongly bias and hide trends that occur within the other projects. This first list of identified GitHub projects that include UML can be found online[7].

Table 1: Found distribution of model files by formats

	xmi	uml	jpeg	png	gif	svg	bmp
Share	3.4%	44.9%	4.7%	29.6%	16.6%	0.6%	0.2%

Results for RQ1: The here identified repositories with UML files represent already 0.28% of the GitHub repositories. Of these, two thirds of the projects contain a single UML file.

5.2 RQ2: Versions of UML models

The next important question was whether models are 'read-only' or also sometimes updated.

Table 2 summarizes the distribution of model files by number of updates per model. Our results show that the vast majority of the UML files (18 867) are never updated. Nonetheless, we found that more than 11% of the UML files in our sample (2 449 models) were updated one or more times. Further, the number of updates of models that are updated is

[5]Replication package http://oss.models-db.com

[6]One of the projects is "eclipse/emf.compare/", which includes more than 6 000 models. We strongly assume that many of these models are generated, e.g. for tests.

Table 2: Distribution of files / projects by number of updates

number of updates	models in projects with 1 to 99 models	models in projects with ≥ 100 models	projects
0	7 947	10 921	2416
1	946	466	332
2	336	42	157
3	151	19	78
4	107	7	64
5	82	2	51
6	67	4	34
7	38	1	18
8	24	3	17
9	24	1	12
10	11	2	8
<20	70	3	50
<30	24	0	25
<40	14	0	8
<50	1	0	6
<60	2	0	2
<70	0	0	0
<80	0	0	2
<90	1	0	3
<100	1	0	1
>100	0	0	11

on average 3,0 times (although the median, which is more significant given the skewed distribution, is 1 time). Furthermore, Table 2 summarize the distribution of projects by sum of model updates or all models of a project.

26.67% of the projects in our sample include at least one model update. Models are less often updated in projects that have more than 100 models (38.09% in our sample), in contrast to 26.60% of the models in projects with less than 100 models are updated. There are only 11 projects that include more than 100 model updates.

Results for Q2: Only 26% of the investigated projects updated their UML files at least once.

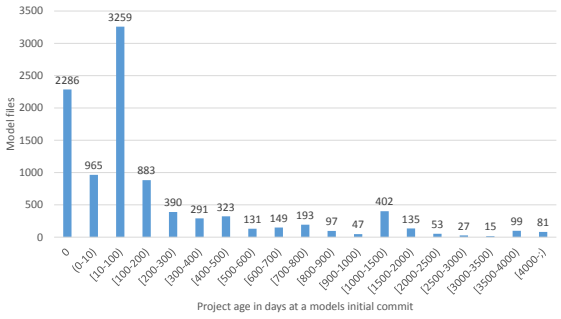


Figure 4: Distribution of model files sorted by project's age in days when the diagram was introduced (models within projects that have less than 100 models)

5.3 RQ3: Time of UML model introduction

Figure 4 shows the dates of the introduction models considering the amount of days since the start of the project, while Figure 5 displays the same information by dividing the duration of the project from the start to nowadays in a normalized way (so, the 50% mark would be half of the project duration since its start until today).

Projects with less than 100 UML models seem to have a tendency to introduce models at the project start. In contrast, the 21 projects with 100 or more models show a different graph. We decided to show the numbers separately, since these projects with partially more than 1 000 models would easily bias the presented view.

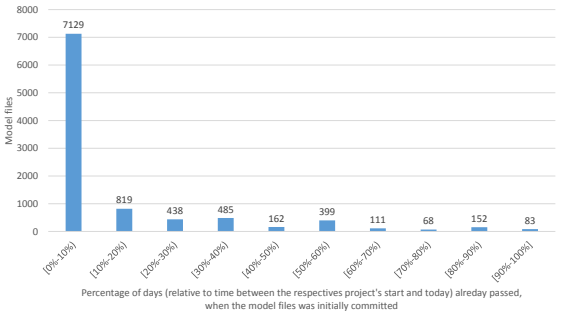


Figure 5: Distribution of model files sorted by percentage of project time that passed when the UML file was introduced (for projects that have less than 100 models)

However, we found that calender time (days) may not be the best way to consider a project's progress, since the amount of activities can highly vary during the lifetime of open source projects. Figure 6 shows the distribution of the models based on the time of their introduction when measured by the percentage of the project's commits.

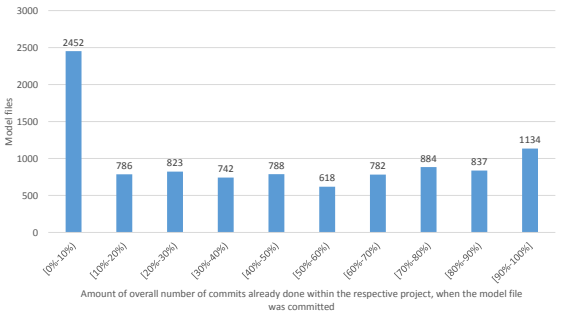


Figure 6: Distribution of files sorted by number of overall commits done when the diagram was introduced (For models in projects that have less than 100 models)

An interesting difference between the two views is that the consideration of time in terms of amount of commits shows a much more balanced view. While this may not be the most intuitive notion, it helps to place the modeling activities relative to the active phases of the project. Thus we can see whether model introduction happened before or after a majority of other development activities (such as coding or documenting). In addition, it helps to better represent projects that had their main activity in the past and/or have become inactive. From our results, it can be seen that new

models are introduced predominantly in the early phases (above 25% of them in the first 10% of the commits), but that new UML models are introduced in later phases too.

Finally, as mentioned above the results look very different for the 21 projects that have 100 or more models. As Figure 7 illustrates there most models are introduce during the last third of the project activities.

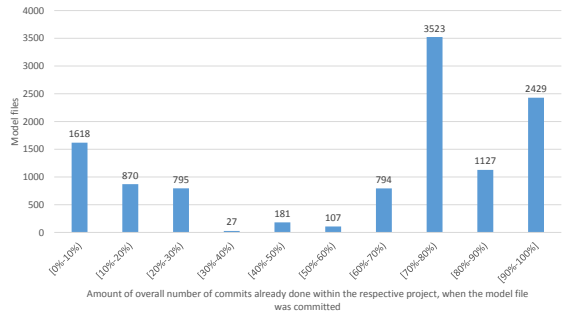


Figure 7: Distribution of files sorted by number of overall commits done when the diagram was introduced (For models of the 21 projects that have 100 or more models)

Results for Q3: UML models are introduced in all active phases of a project with a tendency towards the early phases.

5.4 RQ4: Time span of active UML

In this RQ, we have looked at the time span of active UML creation and modification, i.e., the time between the first introduction of a UML file and the last introduction or update of a UML file within a project.

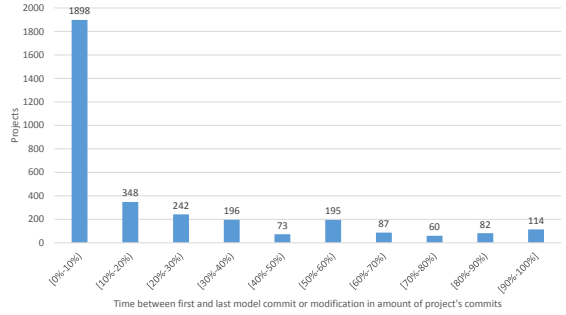


Figure 8: Projects by time between first model commit and last model update-or-commit as percentage of project's commits

Figure 8 summarizes these time spans. The maximum time span found is thereby 100% of the projects commits, while the median of the time spans is 5.8%. We found that by far most projects seem to introduce (and update) all models within a single day. Model creation and updating plays only in a minority of the projects a role during more than 10% of the project's commits.

As with RQ3, we use commits as an alternative measure of the time where UML introductions/updates occur. Figure 9 presents the active UML phase for all 3 295 projects from this perspective. The active UML phase of a project is given horizontally in percentages of commits done, starting when the first model is introduced and ending when the last model is introduced or updated. The diagram illustrates the above finding that a minority of projects (less than 10%) have UML active phases that cover nearly the whole project life time. For a majority of projects the active UML phase is very short and often concentrated in the first commits.

Results for Q4: Few of the studied projects are active with UML during their whole lifetime. In general, the projects work very shortly on UML, usually at the beginning.

5.5 RQ5: Duplicates

Our final question was whether the 21 316 found model files are all distinct originals. To answer the question we used automated duplicate detection, as indicated above.

As a result we identified that 16 576 of the 21 316 found models where unique in our sample. The remaining 4 741 model files represent 2 300 models of which each occurs at least twice. Thus, 21 316 found model files include together 18 876 distinct models. In Figure 10 we summarize how often models with duplicates occurred in our sample. Interestingly, one of the models was found 79 times. In average, models (if duplicated) are duplicated 3.63 times.

Furthermore, we investigated, whether model duplicates belong to the same project. To our surprise this is the case only for the half of the models with duplicates. However, the roughly the half of these models have occurrences in multiple repositories (up to 43). In average the number of projects over which duplicates of a model are spread is 1.88. Figure 11 summarizes the results in form of a histogram.

While duplicates that occur in the same repository might be result of attempts to model versions, we cannot explain the high number of cases were models occur in multiple projects. A possible explanation might be that models might be stored as part of platforms or plug-ins that are reused in multiple projects. Another explanation could be project forks that are done manually by cloning repositories instead of using GitHubs fork mechanism.

Results for Q5: While most models seem to be unique, a large number of identified distinct models (12%) occur several times. In average duplicates are spread over 1.88 projects.

6. DISCUSSION

Considering our initial expectations we were surprised to find such a big number of projects with UML. Surely, 3 295 projects are still a small number compared to the overall number of GitHub projects. Nonetheless, the identification of 21 316 UML models exceeds by far the expectations that we had based on the numbers of models found so far in open source projects in related work, e.g. 121 models by Langer et al.[12] or 19 projects with UML by Ding et al.[4].

Data consistency.

We want to shortly discuss the type of data that we can get with the presented mining method. The method we applied is not trivial and consist of several steps of data

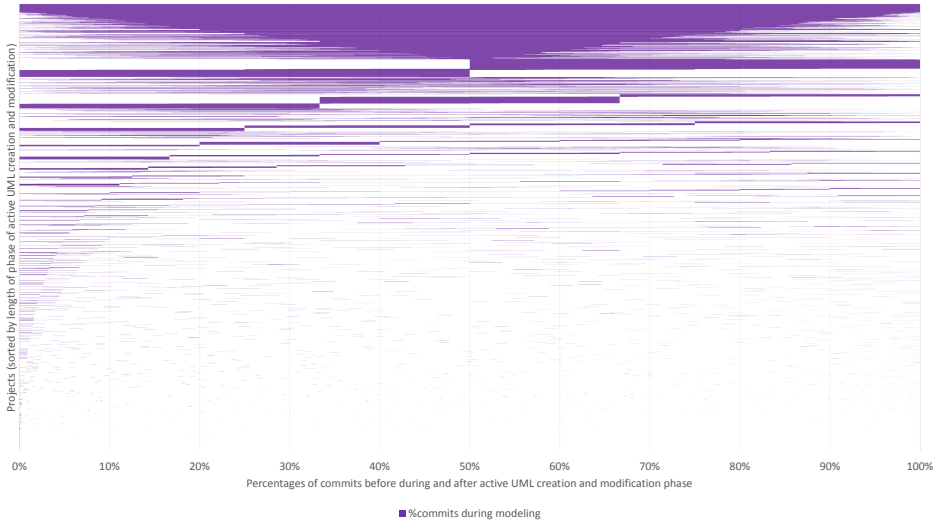


Figure 9: Plot of all 3 295 projects illustrating the placement of active UML creation and manipulation phase within the overall project life span. Time is measured in percentages of commits done, when the first model is introduced and the last model is introduced or updated. The projects are sorted by the relative amount of the active modeling phase (projects with a relatively long active modeling phase are at the top, projects with a shorter phase are at the bottom).

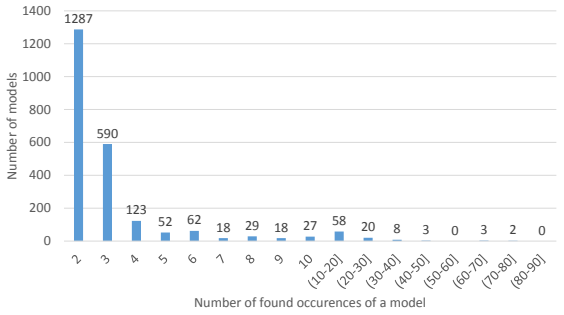


Figure 10: Histogram of models that were found at least twice indicating how often models occur.

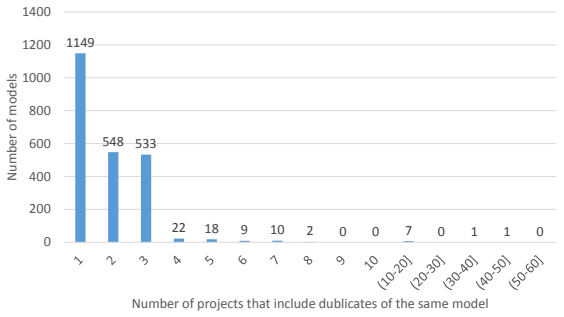


Figure 11: Histogram of models that have duplicates in one or more projects. The histogram shows the number of models by number of projects within which occurrences of a model were identified.

collection. For example, we search for UML candidates using a GHTorrent dump, but accessed the GitHub API to retrieve further information about model contributors. Due to the difference in time between the creation of the GHTorrent dump and the request to the GitHub API, we had drop outs of identified models/projects during the second step.

In addition, we performed this method for the first time, which had an exploratory component in trying out what kind of data we can (and need to) retrieve. This led to the situation that we accessed the GitHub API several times, leading to different drop-outs in models and projects for the different types of information collected.

A *lessons learned* is that, for the next analysis, we have to make a clear planning of all required data in advance, to ensure that at least the second threat to data consistency can be reduced. For this paper we addressed the problem with a reduction of the finally analyzed data set to models and projects for which we had the data points that are necessary to answer the different research questions.

Static models.

A finding is that many projects use UML only in a very static way. In such projects models are never updated and often all models are introduced at the same point in time. These results confirm findings from smaller studies such as Yatani et al.'s [23] or our own (Osman et al.'s [15]), who both found that updates of models are rare. This can have different reasons. One optimistic interpretation would be that models are just introduced as first architectural plans that are followed and used as documentations, but never changed. Another rather pessimistic interpretation would be that modeling is just "tried-out" at some point in time and then dropped. An observation that at least supports the idea that the optimistic interpretation plays a role is that in most projects the main activities of introducing models happen during the first half of all commit activities.

Projects with regular model usage.
Another number that we consider surprisingly high is the number of projects (or models) with more than 20 updates as well as projects with more than 1 year of active UML creation and modeling. Again, compared to the number of overall GitHub projects the here found number seem small. Nonetheless, it was unexpected to find several projects that seem to use modeling on a regular basis.

It has to be noted that the results we found are in contrast to the study of Langer et al. [12] who found an average model lifespan of 1 247 days, while studying 121 enterprise architecture models in open source. We found much lower lifespans. The difference in the findings might be caused by the fact that enterprise architect is a modeling tool that is rather used in an industrial context. Thus, the probability that the projects studied by Langer et al. [12] have industrial support is very high.

Model genesis.
An aspect that we could not address in this study the source of the models or the reason for model usage. Accordingly, the data set was not filtered to exclude for example student projects. We expect this to influence the the findings in this paper, since student projects might show different patterns of model updates, model introduction time, and life span than non-student projects. Addressing this threat will be subject to future work.

Different populations.
A finding that is supported by multiple of the figures shown above is that there seem to be different populations of model usage. A first hint that the data set covers different populations can be seen in Table 2. There is a difference in the number of model updates between projects with more than 100 model files and projects with less than 100 model files. One reason for different populations could be the actual form of model usage and creation. Models might be created manually or automatically (e.g. through reverse engineering). They might solve as plans for system design or as description for an already existing system. Model updates might be performed in order to make small corrections after an initial creation (leading to updates within in short span of time) or in order to make a documentation up to date after a longer phase of system change. At the current state we do not know whether these populations can actually be distinguished on their characteristic commit and update pattern. However, a further hint that they might play a role can be seen in the relatively constant distribution models by the amount of commits that were already done within a project (see Figure 6). We can see model introductions at all project ages. The in average short time of active UML creation and modification speaks against the idea that these introductions at different points in time happen within the same projects. Thus, it seems that we have to deal with different groups of projects introducing their models at different points in time. In future work we plan to have a closer look at the model usage in order to study whether we can associate pattern to different populations of model use.

Duplicates.
The large number of identified duplicates leads to questions. What are the reasons for duplicates? Missing model versioning techniques alone cannot explain the found results. Furthermore, it is not clear yet whether these duplicates represent a form of model use. E.g. if models are adopted together with code from other projects, they might be used to understand the alien code that is embedded in a new project.

Paving the way for future research.
Finally, one of our main contributions is that we presented a method to systematically mine for UML models in GitHub and that this leads to an enormously promising set (much larger than any existing set of projects) for future analysis. On the one hand this will help us to address in future question that arise from the findings of this paper. For example, concerning the model updates, it would be interesting to consider following questions:

- Are models updated by their original authors or by other people?
- In how many projects are UML files obsolete?

Further considering the time of model introduction, we would like to address the following question further: Has the time of introduction an influence on the "success" of an open source project, i.e. the question how many developers join a project? And of course we would like to address the question whether different populations of model-usages can be statistically distinguished.

Even more important, the hereby published list of open source projects using UML can help other researchers to progress in their studies. For example:

- What kind of UML diagrams are used most often?
- What coding languages are used most often in combination with UML?
- What files are changed together with changes in architectural models?
- Can UML help to attract and integrate inexperienced developers?

Furthermore, the data can be used to find case studies for other model or architecture related research, such as:

- Does a good architectural design in models help to create a good architecture in the code?
- Tools for traceability management and model merging can benefit from the real case studies.
- Research that integrates models into fault prediction can be evaluated with the help of that data.

Thus, we believe that the identified initial list of open source projects with UML will be of great help for other researchers, too.

7. THREATS TO VALIDITY

We defined a number of threats to our research's validity. We categorized them by using the validity terminology introduced by Wohlin et.al [22]. We identified three types of threats to validity, they are: Construction Validity, External validity and Conclusion Validity.

7.1 Threats to construct validity

There were a number of threats that might cause the loss of UML files during data collection phase:

- With regards to the materials that were used to collect data, we used a subset of GHTorrent SQL dump from 2015-06-18 which is out-dated at the current time. Accordingly, newer projects have a higher probability to be dropped out. In addition, the limitation of 5 000 hits per hour of GitHub API made data collection last long. Requests that were done at different points of time during the period could give different outcomes, and probably the loss of potential UML files.

- Our collection method, which made use of a number of heuristic filters, might overlook potential UML files which are not complying with searching terms and file-type list. We noticed some cases where UML files had been named differently such as *act-cartesortir.jpg* and *FrameworkInterface.png*. Further, we restricted the search to file formats for which we had techniques to decided, whether the file includes UML. This excludes a couple of other formats which might include models, such as some formats from modeling or graphic tools (e.g. visio files or enterprise architect files), but also documents that might include models as part of documentations, e.g. pdf and word (docx) files or powerpoint.

The loss of UML files might affect to our analysis in the sense that it could make us underestimate the number of projects with UML models and the number of UML models. Being aware of the above consequences, in this research, we don't use our data to analyze the frequencies of model usage as well as the evolution of model usage in general over *the years*. We were focused on getting an overview of various aspects of the use of UML in GitHub projects. We expect no systematic bias concerning the aspects that we investigated!

The applied mechanisms for duplicate detection allow us to identify duplicates within the same file type. However, we cannot identify whether an image and an .xmi file are duplicates. This might lead to an underestimation of the amount of models in this paper. Despite this limitation, our results are already interesting and we consider them a valuable staring point, towards a better understanding of model usage in FOSS.

Kalliamvakou et.al discuss a number of promises and potential risks that researcher might be faced when mining GitHub repository [10]. We found that the threat that many active projects might not conduct all their software development in GitHub could somehow mitigate our analysis.

7.2 Threats to external validity

During data collection phase, in order to minimize the possibility of incorrectly collect non-UML files, we excluded some tool-specific file types form the search for UML models. This might reduce the generalization of our results with respect to these UML tools. However, most of these tools, e.g. Enterprise Architect, are commercial. It is to be investigated in future work whether they are used in open source projects to a similar degree as non-commercial formats.

Data in this research was only taken from GitHub, but not other OSS hosts/platforms such as SourceForge, Google Code, etc. As they differ to each other in terms of size, functionality, users and user's behaviors, the results of this paper can hardly be generalized to the other platforms. It is possible that UML is used in a different ration within projects at other platforms. However, as GitHub is one of the biggest player in the field, we strongly believe that our investigation gives valuable insights to a majority of the OSS community.

A manual glance at the retrieved list of UML models shows that several project paths include names such as "Assignment" or "master's thesis". While this is no direct threat to our results, it limits the generalizability. For example, it is possible that many of the projects that include single UML files only, actually are result of university teaching.

Last but not least, outcomes of this research can not be generalized to closed source community.

7.3 Threats to conclusion validity

As described above, the data has some limitations which permit to do analysis of frequencies, since we expect to have only discovered a part of the overall set of UML models and respective projects. In particular we have not considered powerpoint, pdf, and word-formats of documentation in which UML models may be embedded. For that reason we do not do statistical analysis or even predictions, but stay on a descriptive level in this paper. Nonetheless, we are convinced that this descriptive analysis already represents a valuable contribution to the research community.

8. CONCLUSIONS

In this paper we joined forces in repository mining and model identification in order to identify open source projects on GitHub that contain UML models.

As a result we can present a list of 3 295 open source projects which include together 21 316 UML models. This is the first time the modeling community can establish a corpus comparable to collections already exist for source code only, such as QualitasCorpus [7]. Furthermore, the relatively low amount of UML projects amongst the investigated GitHub projects (0.28%) reconfirmed that our systematic mining approach was required in order to establish the corpus.

We analyzed the data to gain first descriptive results on UML model usage in open source. One finding is that the majority of models is never updates, but that projects exist that do update their models regularly. Furthermore, we learned that models can be introduced during all possible phases in the lifespan of an open source project. Nonetheless a peak of model introduction is during the first 10% of the duration of projects.

A few projects are active with UML during their whole lifetime. However, most projects work very shortly actively on UML, usually at the beginning. We found that 12% of the distinct models occurred several times. Duplicates are in average spread across 1.88 projects.

In the future we plan to further explore the possibilities that arise with the here presented new method to collect data about UML usage in open source projects. For example we plan to analyze the impact of model usage on project dynamics, such as the number of people joining projects. We are planning to proceed with mining GitHub in future work. Based on the now investigated 10% of GitHub we expect that GitHub includes around 34 000 projects with UML and together around 200 000 UML models. Furthermore, we will investigate possibilities to identify UML models that are embedded in other files such as manuals stored in pdf.

[7] QualitasCorpus http://qualitascorpus.com/

9. REFERENCES

[1] O. Badreddin, T. C. Lethbridge, and M. Elassar. Modeling practices in open source software. In *Open Source Software: Quality Verification*, pages 127–139. Springer, 2013.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[3] E. Chung, C. Jensen, K. Yatani, V. Kuechler, and K. N. Truong. Sketching and drawing in the design of open source software. In *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*, pages 195–202. IEEE, 2010.

[4] W. Ding, P. Liang, A. Tang, H. Van Vliet, and M. Shahin. How do open source communities document software architecture: An exploratory survey. In *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*, pages 136–145. IEEE, 2014.

[5] R. France, J. Bieman, and B. H. Cheng. Repository for model driven development (remodd). In *Models in Software Engineering*, pages 311–317. Springer, 2007.

[6] G. Gousios and D. Spinellis. Ghtorrent: Github's data from a firehose. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 12–21. IEEE, 2012.

[7] R. Hebig, T. Ho Quang, G. Robles, and M. R. Chaudron. List of identified projects with uml and replication package. http://oss.models-db.com.

[8] T. Ho-Quang, M. R. V. Chaudron, I. Samúelsson, J. Hjaltason, B. Karasneh, and H. 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.

[9] Y. Jiang and B. Adams. Co-evolution of infrastructure and source code - an empirical study. In *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015*, pages 45–55, 2015.

[10] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 92–101, New York, NY, USA, 2014. ACM.

[11] B. Karasneh and M. R. Chaudron. Online img2uml repository: An online repository for uml models. In *EESSMOD@ MoDELS*, pages 61–66, 2013.

[12] P. Langer, T. Mayerhofer, M. Wimmer, and G. Kappel. On the usage of uml: Initial results of analyzing open uml models. In *Modellierung*, volume 19, page 21, 2014.

[13] S. McIntosh, B. Adams, and A. E. Hassan. The evolution of ant build systems. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 42–51. IEEE, 2010.

[14] S. McIntosh, B. Adams, T. H. Nguyen, Y. Kamei, and A. E. Hassan. An empirical study of build maintenance effort. In *Proceedings of the 33rd international conference on software engineering*, pages 141–150. ACM, 2011.

[15] M. H. Osman and M. R. V. Chaudron. UML usage in open source software development : A field study. In *Proceedings of the 3rd International Workshop on Experiences and Empirical Studies in Software Modeling co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2013), Miami, USA, October 1, 2013.*, pages 23–32, 2013.

[16] G. Reggio, M. Leotta, and F. Ricca. Who knows/uses what of the uml: A personal opinion survey. In *Model-Driven Engineering Languages and Systems*, pages 149–165. Springer, 2014.

[17] G. Robles, J. M. González-Barahona, D. Izquierdo-Cortazar, and I. 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.

[18] G. Robles, J. M. Gonzalez-Barahona, and J. J. 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.

[19] H. Störrle, R. Hebig, and A. Knapp. An index for software engineering models. In *International Conference on Model Driven Engineering Languages and Systems (MoDELS) 2014*, pages 36–40, 2014.

[20] M. Torchiano, F. Tomassetti, F. Ricca, A. Tiso, and G. Reggio. Relevance, benefits, and problems of software modelling and model driven techniques - A survey in the italian industry. *Journal of Systems and Software*, 86(8):2110–2126, 2013.

[21] B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens. On the variation and specialisation of workload - a case study of the gnome ecosystem community. *Empirical Software Engineering*, 19(4):955–1008, 2014.

[22] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[23] K. Yatani, E. Chung, C. Jensen, and K. N. Truong. Understanding how and why open source contributors use diagrams in the development of ubuntu. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 995–1004. ACM, 2009.

C.2 An extensive dataset of UML models in GitHub

This paper was published in the MSR conference (Buenos Aires, Argentina), in May 2017. The complete reference is following:

Robles, G., Ho-Quang, T., Hebig, R., Chaudron, M. R., & Fernandez, M. A. (2017, May). An extensive dataset of UML models in GitHub. In Proceedings of the 14th International Conference on Mining Software Repositories (pp. 519-522). IEEE Press.

An extensive dataset of UML models in GitHub

Gregorio Robles*, Truong Ho-Quang†, Regina Hebig†, Michel R.V. Chaudron†, Miguel Angel Fernandez*
*GSyC/LibreSoft, Universidad Rey Juan Carlos, Madrid, Spain
grex@gsyc.urjc.es, mafesan.nsn@gmail.com
†Chalmers — Göteborg University, Göteborg, Sweden
{truongh, hebig, chaudron}@chalmers.se

***Abstract*—The Unified Modeling Language (UML) is widely taught in academia and has good acceptance in industry. However, there is not an ample dataset of UML diagrams publicly available. Our aim is to offer a dataset of UML files, together with meta-data of the software projects where the UML files belong to. Therefore, we have systematically mined over 12 million GitHub projects to find UML files in them. We present a semi-automated approach to collect UML stored in images, .xmi, and .uml files. We offer a dataset with over 93,000 UML diagrams from over 24,000 projects in GitHub.**

***Keywords*-dataset; UML; GitHub; modeling; mining software repositories;**

I. Introduction

The Unified Modeling Language (UML) provides the facility for software engineers to specify, construct, visualize and document the artifacts of a software-intensive system and to facilitate communication of ideas [1]. UML is commonly taught in the computer science curriculum worldwide, and the use of UML is generally accepted in industrial software development.

However, the number of publicly available examples of UML is relatively low. To the knowledge of the authors, the largest UML dataset up-to-date is the one reported in [2], with around 800 UML models obtained by collecting examples from the literature, web searches, and donations. However, that dataset only contains lone-standing diagram. Thus, it cannot be used for studying the software systems and projects associated to these diagrams.

Even though it has been reported the UML is marginally used in Open Source projects [3][1], the large amount of repositories hosted in GitHub offers the possibility to look for a large number of UML models used in software development projects, together with their source code and development meta-data. This is the reason why we have mined GitHub for UML files. The result of this effort is a dataset with over 93,000 files with UML diagrams. These diagrams comprise several types and formats and offer a valuable data source for educational purposes, as they can be used as real-scenario examples in class, and for further research.

The remainder of this paper is structured as follows: Next, we introduce how we have extracted the data. Section III contains the database schema, while section IV offers the possibilities that such a dataset offers to researchers and practitioners. After presenting future improvements in Section V, we detail the limitations and challenges in Section VI. Finally, conclusions are drawn in Section VII

[1]In [3], we used a similar extraction methodology than the one presented here but with only ∼10% of the GHTorrent repositories as of 2016-02-01.

II. Extraction methodology

The data extraction process comprises the following four steps: (i) retrieval of the tree (file list) from GitHub repositories (Section II-A), (ii) identification (grepping) of potential UML files (Section II-B), (iii) automated examination (and manual evaluation) of the existence of UML notation in the obtained files (Section II-C), and (iv) retrieval of the meta-data from those repositories where a UML file has been identified (Section II-D).

A. Step 1: Mining GitHub

We depart with a list of GitHub repositories obtained from GHTorrent [4][2], which offers a list of over 15M non-forked non-deleted repositories. Since GHTorrent now distributes CSV files (one file per table) instead of mysqldump based backups, we use data available in the projects.csv file: the URL of the project and the values of *forked_from* and *deleted* (as we discard those projects that are forks or have been removed/deleted).

For those projects that are not forks nor have been deleted, we retrieve from the GitHub API[3] the tree (file list) for the *master* branch. If the master branch does not exist, then we query again the GitHub API for the branch that the project has set as default, and perform a third request to download its tree. With up to three GitHub calls for each repository, given the GitHub API limitation of 5,000 requests/hour, it would take around 14 months to perform the retrieval of data in this first step. As this would have made the data gathering unfeasible, we downloaded the JSON files in parallel with over 20 active GitHub accounts, which were donated during this process. This reduced the time span to approximately one month. For almost 3 million of the repositories we obtained an empty JSON file or an error message from the GitHub API, because the repository has been removed or made private in the time that goes from GHTorrent obtaining its data (which is before February 1st 2016) and our request to the GitHub API (during Summer of 2017).

[2]Specifically its 2016-02-01 data release: https://ghtstorage.blob.core.windows.net/downloads/mysql-2016-02-01.tar.gz

[3]https://developer.github.com/v3/git/trees/#get-a-tree

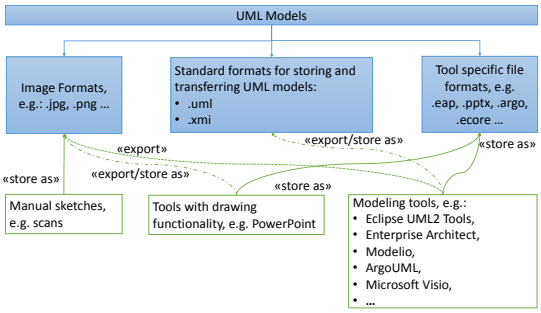


Fig. 1: There is a large variety of tools for creating and formats for storing UML models

Step 1 results in a JSON file per repository with information of the files included in it – altogether, we store around 12,5 million JSON files that suppose 126 GB of compressed data.

B. Step 2: Identify potential UML files

To understand how we searched for files containing UML in the file lists from GitHub, it is important to understand how these files are created and stored. Figure 1 illustrates the different sources of UML files (at the bottom in green). UML models can be created in several ways: (i) by drawing manually, (ii) with the use of tools that have drawing functionality (e.g., Inkscape or Dia), or dedicated modeling tools (e.g., Modelio or Argo UML). Some of the modeling tools even provide the possibility of generating UML models, for example, based on the source code. The variety of tools results in different ways in which UML models can be represented in files. Figure 1 shows these possibilities (at the top in blue):

1) Manual sketches may be digitized with scanners or digital cameras and may be stored in image files of diverse formats.
2) Tools with drawing capabilities may either store the UML models as images, such as .jpeg and .png or .bmp, or may have tool specific formats, e.g. “pptx”.
3) Dedicated modeling tools usually manage file formats that are tool specific, e.g., the “.eap” extension is used by the Enterprise Architect tool. Other tools work with ‘standard’ formats, such as “.uml” and “.xmi”. However, modeling tools with specific formats often allow you to export and import the UML diagrams in these standard formats, as well as in images.

As a consequence, when looking for UML, one needs to consider many different file types. Nonetheless, not all files with a given extension, even those that are tool-specific or standard formats, contain a UML diagram. Therefore, the result of this step will be a list of files that potentially contain UML. These files will have to be checked in the next step.

Given the large amount of files that could be identified as potentially containing UML, we collect in this step only those types of files for which we have automated support to verify that they really contain UML, e.g., we have not considered tool-specific formats and other formats where UML files might be included, such as Word documents (.doc(x)), Portable Document Format documents (.pdf) or PowerPoint slides (.ppt(x)) as there is no current way of extracting the UML models out of them in an automated way.

The list of file types that we look for is composed of:

- Images: Common filenames for UML files (such as “xmi”, “uml”, “diagram”, “architecture”, “design”) that have following extensions (“xml”, “bmp”, “jpg”, “jpeg”, “gif”, “png”, “svg”)
- Standard formats: [“uml”, “xmi”]

The output of step 2 is a list of URLs with potential UML files.

C. Step 3: Verify UML files

Files obtained in the previous step are verified for containing UML diagrams. The procedure followed depends on the nature of the file: images or standard formats. The output of this step is a list of URLs with files with a very high probability of containing UML diagrams.

1) Identify UML images: 423,974 images are successfully downloaded. 18,570 files that cannot be downloaded or opened are removed from the list. 100,032 images that have icon-dimension-size i.e. at most 128 x 128 pixels were excluded.

Some images are icons and duplicates. For them, we i) created a script to automatically detect them; ii) added only one representative image for all duplicates found and iii) marked all duplicate images with the same label obtained in the classification as the representative image. We detect duplicate images with a script that uses the open source .NET “Similar images finder”[4] library. This library offers the degree of similarity between two images by calculating the differences in their RGB projections. Two images are considered similar if the degree is above a given threshold; after several tests on a subset of all images, we used a threshold of 95%.

The final image set of 154,729 images were classified as UML or non-UML images with support of an existing classifier [5]. In particular, all images were first classified as UML class diagrams or non-UML CD images. Then we manually looked for other types of UML diagrams (e.g., sequence diagrams, component diagrams, use cases) within the non-UML CD images. Sketches of UML were counted, as well. It took 6 working days of effort by multiple UML experts to complete the task. As a result, we identified altogether 57,822 UML images/models.

2) Identify UML files among .xmi and .uml files: Both .xmi and .uml files are specific XML (eXtensible Markup Language) formats. By manual checking we found that files with the .uml extension are surely UML. We decided to include all of them in the final UML file list.

The XML Metadata Interchange (XMI) is an Object Management Group (OMG) standard for exchanging metadata information. Each XMI file has to contain a schema that defines the format of the content. Looking at this schema allows us to verify if it is a file containing UML. We have

[4]https://similarimagesfinder.codeplex.com/

TABLE I: Number of UML models per file format

File format	.xmi	.uml	images
Number of models	3,700	32,074	57,822

found that the schema reference is generated in different ways by different tools. For example, we found the following three schema references: "org.omg/UML", "omg.org/spec/UML", and "http://schema.omg.org/spec/UML". Thus, we performed the following identification procedure:

1) Identify possible schema references, by searching for "UML" and "MOF" (the meta-model of the UML language) in a random subset of the models. 7 different references were found.
2) Download .xmi files and parse them for their schema references. We identified 3,700 files with UML schema references.

D. Step 4: Metadata Extraction

The input of this phase is a list of URLs that link to files that contain UML. The GitHub repository where the file is hosted can be identified from its URL. We downloaded all repositories where at least one UML file was identified and extracted its metadata with the help of the `perceval` tool[5], an enhanced version of CVSAnalY [6] that allows to retrieve meta-data from git repositories in parallel.

After this process, we had identified 93,596 UML models from 24,717 repositories. Figure I shows the number of the UML models by their file format. We stored links to these models and their corresponding meta-data in a SQL database. A new SQL table was added to the ones provided by CVSAnalY with just the UML files for easy and efficient querying.

III. Database schema

The main dataset consists of two CSV files [6].

- **UMLFiles_List.csv** lists all identified UML files, sorted by project name.
- **Project_FileTypes.csv** lists all projects with summary information and statistics per project, including the number of identified UML files and the file format (.xmi, .uml, .jpg, .jpeg, .svg, .bmp, .gif, or .png) of the UML files.

The first line in the CSV files contains self-explanatory variable names of the columns. In addition, we provide meta-data of the repositories where UML files have been identified; its database schema is shown in Figure 2. The main entities and relationships are as follows:

- **repos**: Each repository has a unique *name*, a *founder*, a *URL* to its GitHub page, and a total number of commits. Dates of the first commit and the last commit are recorded in *first_commit* and *last_commit*, respectively.
- **umlfiles**: This table contains information of all UML files/models. Each UML file has a unique *id*, and belongs to a specific repository characterized by a *repo_id*. Each UML file has a *name* and can be changed in multiple commits. The *commits_id* field is a foreign key to the *commits* table, where all commits to a given UML file can be tracked. Field *url* shows the URL of the latest (i.e., the current version) UML file on GitHub.
- **commits**: Each commit has a *gh_id* (a.k.a. sha) which is a global unique identifier, a *commit date* and belongs to one *repository*. A commit is committed by one person whose id is *people_id*. The number of files changed within this commit is recorded in *cochanged*.
- **file_commits**: Each file commit has a unique ID and belong to a specific commit. Field *name* indicates the name of the committed file. Field *filetype* shows the type of the committed file based on a predefined classification (e.g., source code, documentation) [7].

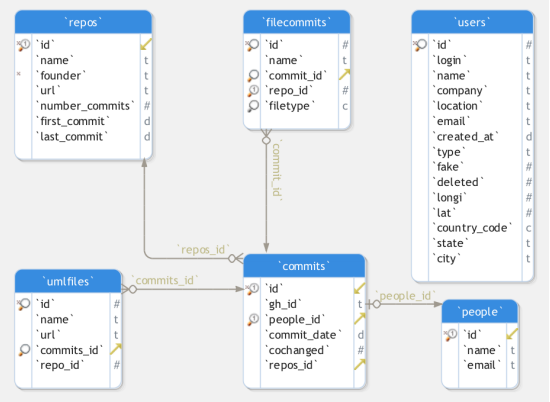


Fig. 2: The relational database schema

Our database schema can be augmented with information provided by GHTorrent (and described in [4]). For instance, in our schema, we have included the *Users* table, which linked with the data in the *people* table (from GHTorrent) which contains demographic information of the committers of UML files.

IV. Research with this data

We consider the dataset as relevant for researchers in the area of software design and modeling, because the whole community lacks good examples of not just models, but software systems that are built with the help of models as well. The need can be seen on several previous initiatives to collect datasets, which are often limited in the number of collected models [7]. The future uses of our dataset can be sorted in three groups:

a) Advantages and Trade-Offs of UML: The dataset can be the basis for empirical studies on the advantages and disadvantages of UML (and modeling). Some researchers have already used it to investigate if anti-patterns are propagated

[5]https://github.com/grimoirelab/perceval
[6]Data-set: http://oss.models-db.com/
[7]Classification of filetypes: https://github.com/MetricsGrimoire/CVSAnalY/blob/master/pycvsanaly2/extensions/FileTypes.py

from models to the code [8]. Our dataset can help to enrich this research, which qualitatively investigates single cases, with quantitative studies. Further, the dataset can help to study in more general how the use of UML modeling impacts the code structure and whether improvements in software quality and productivity can be observed when UML is introduced.

b) UML Use: The dataset can be used to study how UML is used and to develop guidelines for UML novices. For example, the data could be used to learn what model layouts OSS developers use and what average size models have. Studying UML that occurs in images can also deliver hints on needs that OSS developers have for visual highlighting strategies. For example, during the manual check of the images, we have seen a lot of UML images where color was used for highlighting. Furthermore, due to the availability of the models (and the projects they belong to), the dataset will allow to analyze how code and models are related to each other; we still do not know what amount of a software system is typically covered by models and to what degree models abstract the code.

c) Evaluation of Scientific Approaches and Modeling Tools: Constructive research on software modeling often has the problem that there are not enough real cases of models to evaluate newly developed approaches and techniques. Currently, this limitation is worked around on the basis of toy examples or *artificially* generated models. In exceptional cases, researchers are allowed to use obfuscated industrial models or models created with the help of practitioners for the purpose of the evaluation [9]. Our dataset provides real cases of UML models in machine readable form. Professional tool vendors, who provide case tools for modeling, might be able to use the dataset to test new features on real data e.g. layout generation.

V. Future Improvements of the Dataset

Due to the fact that GitHub is a living organism with projects appearing and disappearing over time, it will be necessary to curate the dataset in the future. Especially for models that are stored in images, this is today still associated with manual effort. We believe that image recognition techniques will improve in future and will help automate this task.

Besides that we plan to extend the dataset in the future. For example, we still do not cover all file types that include UML. Similarly, software models that do not follow the UML standard, such as SysML models, are not part of the dataset.

However, it is not just future extensions that will make the dataset more valuable, but also annotations that can be made to the dataset. We have been requested to label the UML diagram types used. Similarly, information about the goals of project for using models, e.g., for design or documentation, can be a valuable addition.

VI. Limitations and Challenges

Although the dataset is a huge progress for research on model driven engineering, there are still some limitations that should be considered when using it.

First, there are general issues with GitHub data, such as the high number of student projects [10]. Many of these problems also hold for our dataset. Researchers using the dataset should filter it beforehand according to their needs.

GitHub being a dynamic environment, it is possible that projects and models become inaccessible over time. Users might experience that single projects or files cannot be found in GitHub anymore. In addition, the dataset is not a complete list of UML in GitHub, e.g., due to limitations in the searched file formats. Therefore, the dataset cannot be used to know the frequency of UML in GitHub projects.

Finally, due to the large scale of the dataset we cannot exclude that some of the files identified as UML are false positives, i.e., do not actually include UML. We have put a lot of effort, e.g., with manual checks, into ensuring the quality of the data. Researchers using the dataset should have a critical look at the models.

VII. Conclusions

We offer a dataset with over 93,000 publicly available UML models in GitHub from over 24,000 projects, offering a dataset that is two orders of magnitude larger than current datasets. Mining GitHub and identifying UML diagrams is not a trivial task. The main challenges that we had to face to obtain the dataset have been because of the large amount of data that we had to handle, and the assessment of the different types of files that are used to store UML. Our dataset offers many possibilities for research and education on UML and modeling.

References

[1] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[2] B. Karasneh and M. R. V. Chaudron, "Online img2uml repository: An online repository for UML models," in *EESSMOD@ MoDELS*, 2013, pp. 61–66.

[3] R. Hebig, T. H. Quang, M. R. V. Chaudron, G. Robles, and M. A. Fernandez, "The quest for Open Source projects that use UML: Mining GitHub," in *Proceedings 19th International Conference on Model Driven Engineering Languages and Systems*, 2016, pp. 173–183.

[4] G. Gousios and D. Spinellis, "Ghtorrent: Github's data from a firehose," in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*. IEEE, 2012, pp. 12–21.

[5] T. Ho-Quang, M. R. V. Chaudron, I. Samúelsson, J. Hjaltason, B. Karasneh, and H. Osman, "Automatic classification of UML class diagrams from images," in *Proceedings of the 2014 21st Asia-Pacific Software Engineering Conference - Volume 01*, 2014, pp. 399–406.

[6] G. Robles, J. M. González-Barahona, D. Izquierdo-Cortazar, and I. Herraiz, "Tools for the study of the usual data sources found in libre software projects," *International Journal of Open Source Software and Processes*, vol. 1, no. 1, pp. 24–45, 2009.

[7] H. Störrle, R. Hebig, and A. Knapp, "An index for software engineering models," in *International Conference on Model Driven Engineering Languages and Systems (MoDELS) 2014*, 2014, pp. 36–40.

[8] B. Karasneh, M. R. V. Chaudron, F. Khomh, and Y.-G. Gueheneuc, "Studying the relation between anti-patterns in design models and in source code," in *Software Analysis, Evolution, and Reengineering (SANER), 23rd International Conference on*, vol. 1, 2016, pp. 36–45.

[9] P. Pietsch, D. Reuling, U. Kelter, J. Folmer, and B. Vogel-Heuser, "Experiences on the quality and availability of test models for model differencing tools," in *Free Models Initiative Workshop Proceedings*, 2014, p. 11.

[10] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 92–101.

C.3 Practices and Perceptions of UML Use in OS Projects

This paper was published in the ICSE (SEIP track) conference (Buenos Aires, Argentina), in May 2017. The complete reference is following:

Ho-Quang, T., Hebig, R., Robles, G., Chaudron, M. R., & Fernandez, M. A. (2017, May). Practices and perceptions of UML use in open source projects. In Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), 2017 IEEE/ACM 39th International Conference on (pp. 203-212). IEEE.

Practices and Perceptions of UML Use in Open Source Projects

Truong Ho-Quang, Regina Hebig, Michel R.V. Chaudron
Chalmers — Göteborg University
Göteborg, Sweden
{*truongh, hebig, chaudron*}*@chalmers.se*

Gregorio Robles, Miguel Angel Fernandez
GSyC/LibreSoft
Universidad Rey Juan Carlos, Madrid, Spain
grex@gsyc.urjc.es, mafesan.nsn@gmail.com

***Abstract*—*Context:* Open source is getting more and more collaborative with industry. At the same time, modeling is today playing a crucial role in development of, e.g., safety critical software. *Goal:* However, there is a lack of research about the use of modeling in open source. Our goal is to shed some light into the motivation and benefits of the use of modeling and its use within project teams. *Method:* In this study, we perform a survey among open source developers. We focus on projects that use the Unified Modeling Language (UML) as a representative for software modeling. *Results:* We receive 485 answers of contributors of 458 different open source projects. *Conclusion:* We found out that collaboration seems to be the most important motivation for using UML. It benefits new contributors and even contributors who do not create models. Teams use UML during communication and planning of joined implementation efforts.**

***Keywords*-UML; architecture documentation; OSS projects; GitHub; motivation; communication; effectiveness of UML**

I. Introduction

Open Source, which has its roots in the free software movement, started partially as a counter-movement to the software industry in the 80s and 90s [1]. Even though, there was a clear border between Open Source and industry, in the late 90s and early 2000s, the situation started to change. In those years, some industry started to early adopt the open source movement practices, collaborating with communities [3], or some companies were created around some communities [4]. Many projects created foundations to serve as an umbrella to collaborate and integrate software industry partners [5].

Thus, we have witnessed a process and technology transfer among open source software (OSS) and industry, that has made the line between both be vague nowadays. Notable contributions from OSS to industry have been technologies, such as git and GitHub, and community-managing practices, although the list of adoptions is much larger [6]. On the other hand, OSS has embraced practices from industry, such as (modern) code review practices and planning and requirements analysis mechanisms [7]. Companies with a large pool of developers try to have an "internal" OSS-like ecosystem, a concept coined as inner source [8]. Many OSS practices are commonly taught at universities and young graduates start their professional careers with experience in OSS, whether in languages (Python, Perl, Ruby...), products (JQuery, Hadoop...) and tools (GCC compiler tool chain, git and GitHub...) [9]. And the software industry is looking into popular Open Source repositories, such as GitHub, to find suitable candidates to fill open development positions [10].

In this regard, we have seen a clash of two worlds, resulting in new practices where industry sometimes has adopted elements from OSS and vice versa. As the trend seems to go on, we would like to draw attention on modeling, specifically on the use of Unified Modeling Language (UML) in OSS. UML has been around as a graphical language for modeling software systems for about 25 years. As far as it is known, UML is not yet frequently used in OSS projects, with a rather marginal use [11]. OSS is known to be programming-driven, with other tasks having room for further improvement [12]. However, modeling is used in major companies [14]. Modeling is, thus, an area where we can find a gap between OSS and industry. Given that the use of UML in OSS is not very well-known, we would like to shed some light into this issue with the aim of discovering how UML is used and whether it is considered useful. We hope that the results will help to understand whether the use of UML in open source helps these projects and whether industry working with open source projects should promote the UML use.

To this end, we used a technique, that we developed to find UML in GitHub projects [11]. The paper showed the feasibility of our approach and triggered us to come up with various research questions addressed in this paper. For this paper we scanned through the majority of non-forked GitHub projects (over 12 million of projects) and identified which of these projects use UML.

We performed a large scale survey directed at those projects that use UML, with focus on how the UML is used and how it impacts development activities. Contributions of this research are: i) the identification of a large set of open source projects that use UML and ii) insights from a large scale survey under developers of open source projects into use of UML. Amongst other insights, we found that UML is used to coordinate the development. Furthermore the use of UML seems to help new contributors to get started, while it does not seem to attract new contributors. The set of projects we identify are a valuable resource for future empirical studies regarding the UML.

The rest of this paper is constructed as follow: We formulate a number of research questions in Section II, then introduce related work (Section III) and describe our research method in Section IV. Section V presents our findings. We discuss the our findings, possible threats to validity and implications of our research in Section VI and giving conclusions in Section VII.

II. Research Question

To better understand the use of UML in OSS, we formulate the following 3 main research questions:

RQ1 *Why is UML used in OSS projects?*

To get an impression of the role of UML models in OSS we formulate following this first question.

- $SQ_{1.1}$ What are the motivations to use UML modeling?
- $SQ_{1.2}$ What are the reasons not to use UML in projects?

RQ2 *Is UML part of the interaction of (a team of) contributors?*

Teams and interaction between developers play an important role within todays software intensive industry [13]. Models are used as artifacts that are basis for planning and work coordination. However, it is an open question, whether UML models fulfill a similar role in OSS projects. We approach this question from three aspects: 1) the awareness within the project that UML models are available, 2) the use of UML during project planning and communication, and 3) the role of UML during joined implementation efforts. These three sub-research questions are structured:

- $SQ_{2.1}$ Are developers aware of the existence of UML in their projects?
- $SQ_{2.2}$ Are UML models used during communication and team decision making?
- $SQ_{2.3}$ Are modeled designs adopted afterward during the implementation phase by teams of OSS contributors?

RQ3 *What is impact/benefit of UML?* Much research was performed to identify benefits of UML usage in industry. However, it is not yet clear whether UML usage impacts or even benefits development in open source. Again, we consider three different perspectives: 1) the role of UML for novice contributors, 2) the impact of UML on the working routine, and 3) the impact of UML on the attractiveness of a project for potentially new contributors. The following sub-research questions are structured:

- $SQ_{3.1}$ Can UML models support new contributors?
- $SQ_{3.2}$ What are the impacts of using UML in OSS projects?
- $SQ_{3.3}$ Can UML models help to attract new contributors?

III. Related work

In the following we discuss related studies about UML or modeling in industry and open source.

A. Modeling in Industry

Modeling is widely studied in industry, for example in surveys such as the ones performed by Torchiano et al. [14], Gorschek et al. [15], or Forward et al. [16]. Torchiano et al. found that models help to improve design and documentation. However, they also found that model usage is connected to extra effort, especially due to a lack of supporting tooling. Forward et al. find that models are primarily used for design and documentation, while code generation is rather seldom. Gorschek et al. [15] focused on a different population, which are programmers, partially working in industry and open source. Within their sample design models are not use very extensively. However, models and UML are found to be used mainly for communication purposes. Further, they report on a higher use of models for less experienced programmers.

Besides these big surveys also case studies were performed in order to investigate the impact of the modeling/UML usage. For example, Baker et al. [17] found an increase of productivity when using UML in Motorola. Also Nugroho et al. [18] investigated an industrial case study and found that UML usage has the potential to reduce the defect density and, thus, increase the quality of software. Just as in the case described by Kuhn et al. [13], most of the case studies draw a picture of model use, where models are actually artifacts that are produced and consumed by different people.

B. Modeling in Open Source

Much less work has been done on UML use in open source software. One reason for this is the challenge to actually find cases that can be studied. For example, Badreddin et al. studied 20 projects, without finding UML and concluded that it is barely used in open source [19]. Similarly, Ding et al. [20] found only 19 projects with UML when manually studying 2000 open source projects. However, in our previous work [11] we presented an approach to solve the problem of finding projects with UML by mining GitHub for projects with UML.There are several investigations of single or very small numbers of cases of open source projects that use UML, e.g. by Yatani et al. [21], who found that models are used to describe system designs, but are rarely updated. Osman et al. [22] studied to what ration classes in the diagrams are implemented in the code. Finally, Kazman et al. [23] investigate the Hadoop Distributed File System to learn how documentation impacts communication and commit behavior in the open source system. There are some studies that approach model use in open source with a quantitative perspective, studying large numbers of projects. For example, to study the use of sketches, Chung et al. [24] collected insights from 230 persons contributing to 40 open source projects. Finally, Langer et al. [25] studied the lifespan of 121 enterprise architect models in open source projects.

However, to the best of our knowledge there is so far no quantitative study targeting the use of UML within the team communication and its effects.

IV. Research Methodology

In this section, we describe our study method in details. The overall process is shown in Fig. 1.

A. Data Collection

First step is to identify UML files in GitHub repositories. In our previous work, we were able to identify UML files in around 1.2 million GitHub repositories [11]. In this study, we extend the data collection work to the whole GitHub database using data collection method described in the paper. A number of changes had been made in order to adapt the big data retrieval. In this section, we briefly summarize the data collection steps and the changes that were made.

1) Obtaining the full list of GitHub projects: To obtain the list of projects, we used the data from the February 1st 2015 dump of GHTorrent [26]. From this dataset we could obtain a list of projects that were not deleted and non-forks. However, GHTorrent does not contain information on the files contained in the repositories. Hence, we ended up using GitHub API to obtain the list of files for a total number of 12 847 555 repositories. The result is a JSON file per repository with information on the files hosted in the *master* (or *default*) branch of the repository.

2) Identifying UML files: Next step was to identify UML files from the file list. Firstly, potential UML files were collected using several heuristic filters based on the creation and storage nature of UML files. After that, an automated process was applied to examine the existence of UML notation in the obtained files. A manual validation was made to consolidate the identification result. Details about the identification steps are described in Section 4 of the paper [11]. At the end of this step, we were able to identify 93 648 UML files from 24 797 repositories.

3) Extracting meta-data: For all projects that contain a UML file, development meta-data from the repositories has been retrieved. Therefore, we use `perceval`, an evolution of the well-known `CVSanalY` software [28], that allows to obtain these data in JSON files, making it possible to perform the process in parallel. It took the five instances of the tool over 4 weeks to complete this task. At the end, and after removing 240 JSON files that contained a 404 Not Found response, we had 24 125 JSON files that were parsed and normalized, and finally converted into SQL.

B. Filtering the obtained projects and contributors

In this phase, we aimed at mitigating a number of known threats to validity when mining GitHub database, i.e. sample/short-time projects [29] or identification of contributors [30]. Section IV-B1 and section IV-B2 show the criteria that we applied to filter out short-time projects and to merge duplicate contributors, respectively.

1) Filtering short-time projects: For this paper we aim at projects that are interesting from an industry perspective. Thus, we focus on projects that are not short-term and that do not jut consist of a singe contributor. We define short-time projects as those projects that have: i) active time (time between the first and the latest commits) less then 6 months, OR ii) less than 2 contributors, OR iii) less than 10 commits. After classifying and filtering short-time projects, 4 650 UML-projects (out of 24 125, we use the term *UML project* to refer to GitHub projects that contain UML file(s)) and 2 701 (out of 17 101) non-UML projects met our requirements. The final list of the projects is shared in our replication package[1].

2) Merging duplicate contributors: One contributor can use different emails, user-name (for example, one changed his profile user-name or email during the project time). This might cause duplicate contributors, and as a consequence, projects with one contributor only could be classified as having more than one contributor [30]. We merge identities with different ids that have: i) same e-mail address, or the same full name -in the case of the full name; ii) names have to have at least two words or including a number if only a word, e.g., ”arg123”. This is a rather conservative approach, but it minimizes the number of false positives [30]. After running the script, the original 129 276 contributors result in 99 319 distinct ones.

C. Conducting survey

In the following we give a short overview about how we conducted the survey.

1) Participant: To ensure that be get a balanced picture, we had to consider the role that the different contributors play within the OSS projects with UML. Two dimensions of roles are important (each questioned person would fulfill a combination of roles in these two dimensions):

- Founder (F) vs. non-founder (NF)
- Non-UML Contributor (NUC) vs. first UML Contributor (1UC) & UML Contributor or updater (non-1st contributor) (UC)

Consequently, each interview participant fulfills one of the following 6 roles: F-1UC, F-NUC, F-UC, NF-1UC, NF-NUC, NF-UC. For each project, we randomly selected 3 contributors, to whom we sent the questionnaire. The selected 3 contributors had to fulfill one of the following 3 constellations of roles.

- F-NUC, NF-1UC, NF-UC
- F-1UC, NF-UC, NF-NUC
- F-UC, NF-1UC, NF-NUC

For some projects not all roles could be identified (e.g. there are not necessarily NUCs or UCs). In this case we contacted for respectively less contributors.

2) Questionnaire: The questionnaire is designed to meet the following requirements:

[1]The replication package for this paper can be found at http://oss.models-db.com/2017-icse-seip-uml/

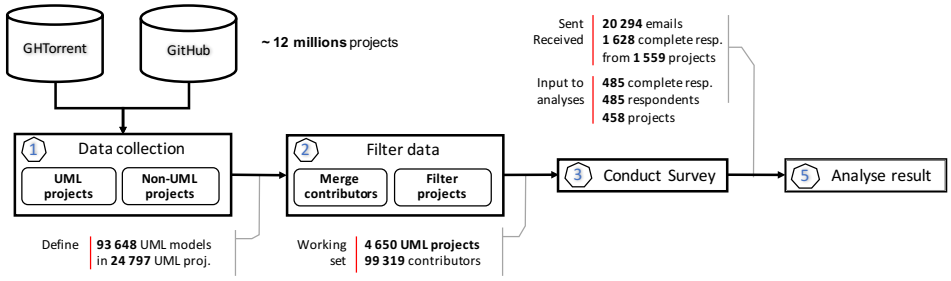


Figure 1: Overall process

Multiple roles: We distinguish for the roles, what questions they get, in order to facilitate different information needs. For example, we would ask NUCs whether they are aware that UML models exist in the project, while we would ask UCs, whether they think that NUCs are aware of it. Thus, depending on the role, participants received between 5 (NF-NUC) and 19 (F-1UC) questions.

Exploration: We use a funneling approach (from broad to narrow) when designing the survey. For example, if a UC uses a UML model for architecture/design purpose, we would ask if the model is adopted, and eventually, who implemented the model. Accordingly, the number of questions would not just be different between different roles, but also between respondents who have the same role. In addition, to gain more insights, we use a mix of close-ended and open-ended questions in the survey.

Personalized Contact: To ensure that participants know what projects and UML models we are referring to, we personalized the email with which we contacted potential participants by concretely referring to his/her GitHub identification, the name of project of interest and (if applicable) an URL to his (first) UML commit or to a UML file committed by someone else. By following the URL (e.g. https://github.com/rvs-fluid-it/wizard-in-a-box/blob/master/src/doc/wizard-in-a-box-design.png), participants could get further contextual information about the UML models, for example commit messages, commit date, etc.

We used Lime Survey tool [2] as the tool provide functionality to fulfill the survey requirement. Our Lime Survey server is hosted at http://survey.models-db.com/. Details about survey settings and email templates can be found in the replication package.

3) Sending out the survey: We sent 20 294 survey emails to OSS contributors in 6 days, from July 21 to July 26, 2016. More than 1 000 emails were not sent because of various problems, including out-dated emails, etc. We sent reminder emails after one week and finally closed the survey in August 4, 2016. Altogether, we received 2 230 responses, of them, 1 628 were completed. After filtering responses that belong to short-term projects, we gained 485 survey responses of respondents from 458 projects.

[2]LimeSurvey homepage: https://www.limesurvey.org/

Table I: Number of emails sent, number of responses and number of responses after filtering by participant categories

	Founder			Non-Founder			SUM
	1UC	*NUC*	*UC*	*1UC*	*NUC*	*UC*	
Sent emails	4509	3891	713	6737	3221	1223	20294
#full resp.	373	293	68	564	210	120	1628
#inc. resp.	167	105	24	214	56	36	602
#fil. resp.	*84*	*79*	*27*	*176*	*80*	*39*	*485*
Percent(%)	17.3	16.3	5.6	36.3	16.5	8.0	100

D. Data Analysis

First, we take into account completed responses only. Second, we do not consider short-time projects.

Part of the questionnaire are free-text questions. We use these questions to learn about phenomena for which we do not know a fixed set of answers, yet. The goal of analyzing these data is to identify reoccurring themes. Therefore, we used a coding technique, following the constant comparison method as described by Seaman [31]. We decided to use an empty starting set of codes and develop them during the coding. For each of the question two of the authors coded the answers independently. In a second step we inspected the codes together to identify and if necessary resolve differences in the selected codes and application of the coding. Afterward, we went a second time through the data in order to ensure that the now fixed set of codes was assigned consistently. We did this i) to increase the quality of the coding and ii) to decrease the probability that we miss interesting aspects. As a final step we checked whether codes occurred for more than one project, in order to prioritize those themes that are of greater relevance.

Furthermore, we took those cases where we got multiple responses for the same project and aggregated them. This aggregation was done as follows: we interpret observation based questions (i.e. whether UML is used for communication) as reports about a project. Thus, aggregating a ”yes” and a ”no” answer for the same project to a ”yes” to indicate that there is a report about a phenomenon for that project. Similarly, we prioritized ”no” over ”I have no opinion”. ”I do” and ”I have seen other people doing” are merged to an ”I do”.

V. Results/Findings

A. Respondent Demographics

A total of 2 230 respondents from 91 countries began the survey, with 1 628 completed compulsory questions of the survey. After classifying survey responses by short-time projects, we ended up with 485 survey responses of respondents from 458 projects. Among 485 respondents, 190 (about 40%) are founders of an OSS project, 159 (32.8%%) are non-UML contributors (Table I). Regarding educational background (as shown in Fig. 2), 37.73% respondents had a Master's degree, 30.31% had a Bachelors, 16,29% had a Ph.D., and 11.75% were still in education. About 4% of the respondents identified as autodidacts. A vast majority of the respondents reported to be familiar with architecture documentation in different formats, leading by UML (90.31%), then auto-generated code documentation and software models in generic formats (78%) (Fig. 3). Only a half of them (45%) were familiar with architectural notations on white papers. There are programming languages where UML is more frequently found (Smalltalk, Java, C# and C++). On the other side, UML has not that much impact among the Objective-C and Ruby community.

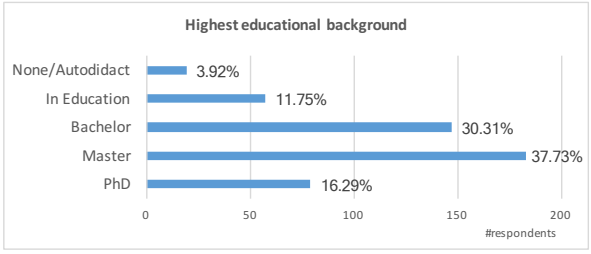


Figure 2: Distribution of respondents based on their highest educational background

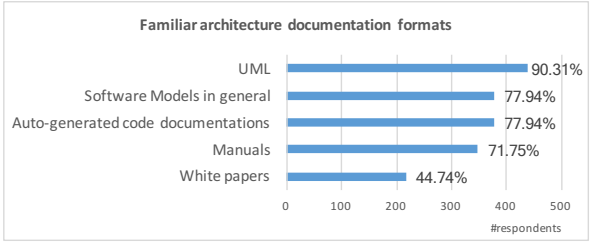


Figure 3: Architecture document formats that respondents were familiar with. The respondents could choose multiple formats

B. Why is UML used?

1) What are the motivations to use UML modeling?:

Fig. 4 shows the answers from 326 UCs (of 319 projects) about the *intent* of UML files they added/updated. Most of UML files served for design/architecture and documentation purposes, with 70% and 71% of votes, respectively. For about 18% of the projects software verification was mentioned as one of the main purposes. Refactoring and code generation less usual (in 14.11% and 12.85% of the projects).

Among 125 NUCs that claimed to be aware of the existence of UML models, 109 people (from 109 projects) reported to find UML helpful. Fig. 5 presents their answers. 79% of the respondents found UML useful for understanding the OSS systems. They also found UML models helpful as the models assisted in improving communication within the project, guiding implementation and managing quality of the project.

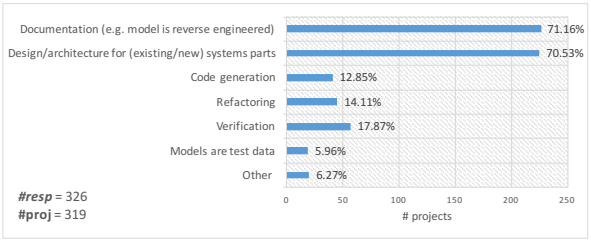


Figure 4: Intent of UML models that were added/updated

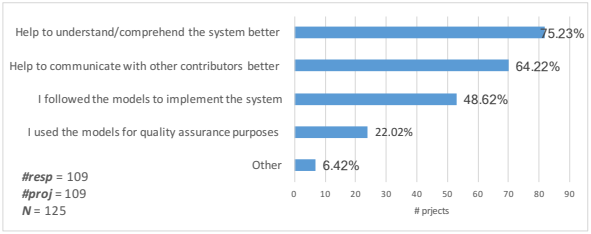
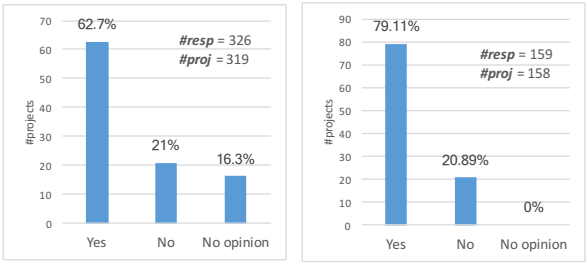


Figure 5: How did UML help non-UML contributors?

Results for $SQ_{1.1}$*: The majority of models are intended for creating software designs and documenting software systems. NUCs use benefit from UML models when it comes to understanding a system and communication.*

2) What are the reasons not to use UML in projects?: To complement our finding on the motivations to introduce/use UML, we asked those 16 NUCs, who did not find UML models useful, for the reasons why. Respondents in 6 projects actually didn't use the models, finding themselves not required to learn/use UML (e.g. *"there was no demand to do so"*). Interestingly, in no case license problems for modeling tools were a problem.

In 4 cases the UML files were outdated. Other reasons that were brought up in free-texts are: missing support for versioning models, a failed attempt to understand the models, a preference for other means of communication (face to face), a preference for other forms of modeling/sketching, a preference for reading code rather than spending time for UML models, and the dislike of UML (anti-UML attitude).



(a) Do UCs think that other contributors are aware of UML?

(b) Are NUCs aware of the existence of the UML models?

Figure 6: Developer's awareness about the existence of UML in their projects (by project)

Results for $SQ_{1.2}$*: Only a small number of respondents found UML not useful.*

C. Is UML part of the interaction of contributors?

1) Developer's awareness about the existence of UML in their projects:
To answer this question we first asked creators/maintainers of UML models whether they think that the models are known by developers of the projects (summarized in Fig. 6a). In 62.7% of 319 projects with responses, the UCs/1UCs believed that UML models are known by developers of the projects. Second, we asked NUCs of projects that use UML if they aware of the existence of UML models in their projects (Fig. 6b). Surprisingly, in the vast majority of the projects (80%) NUCs stated that they are aware of UML models.

To better understand the difference between the answers of UCs and NUCs, we looked in detail into the 24 projects for which we received responses from NUCs and UCs. In 10 out of 24 projects, NUCs and UCs differed. Interestingly, it is in 8 projects the case that UC(s) did not expect their UML to be known by other developers, while the NUCs were aware of it. It seems that model creators tend to underestimate the spread of their models.

Results for $SQ_{2.1}$*: A majority of non-UML contributors are aware of the UML models in their projects. This is even slightly more than expected by the authors of the models.*

2) Are UML models used during communication and team decision making?:
In a first step we asked founders and UCs whether UML models are considered in the communication between contributors. Fig. 7 summarizes the 405 individual responses from 388 projects. According to the responses, UML models were considered in communications in a large majority of the participated projects (60%).

As a step further, we asked whether UML models were used as a basis for architectural decision making or mentoring activities. Respondents from the majority of the projects recalled that they had used the UML models for making architecture decisions (58.7%) and explaining each other different aspects of the system (58.25%) (summarized in Fig. 8).

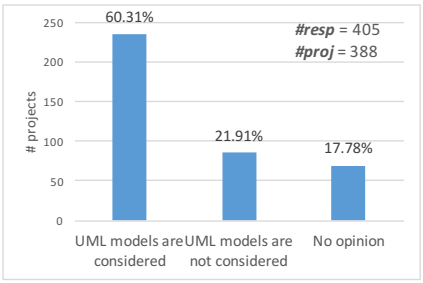


Figure 7: Are the UML model(s) considered in the communication between contributors? (per project)

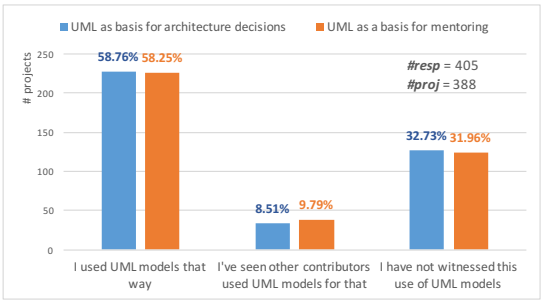


Figure 8: Is UML a basis for architectural decisions or mentoring activities? (per project)

Results for $SQ_{2.2}$*: UML models were considered as a mean of communication and as a basis for architectural decisions and for mentoring in a majority of projects.*

3) Are modeled designs adopted afterward during the implementation phase by teams of OSS contributors?:
For those projects that claimed to have design models, we asked the question "Was the UML models adopted during the implementation phase?". Fig. 9 summarizes for 225 projects the answers of the 231 respondents. In most cases UML models were adopted partly or completely during the implementation phase (about 92%).

If the answers was that UML models were at least partially adopted, we asked further questions to find out who and how many contributors implemented the modeled designs. Fig. 10 and Fig. 11 summarise the responses per project (based on 214 individual responses for 208 projects).

Creators of UML models are greatly involved in implementing the modeled designs (in 88.5% of the projects). Experienced contributors helped in 35.5% of the cases and novice contributors helped in around 13% of the cases.

In the majority of the projects (around 66%) more that 1 person participated in the implementation of previously

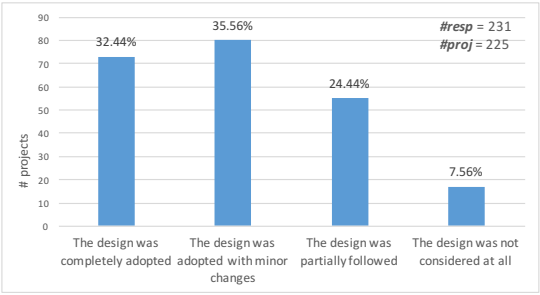


Figure 9: Was the UML models adopted during the implementation phase? (per project)

modeled designs. However, only 7% of the projects reported to have more than 5 contributors involved such joined implementation efforts.

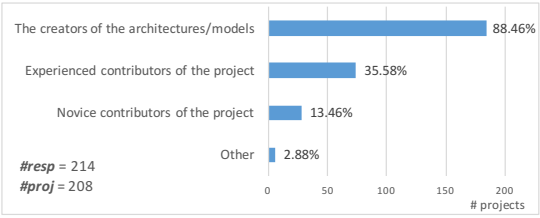


Figure 10: Who implemented the UML models? (by project)

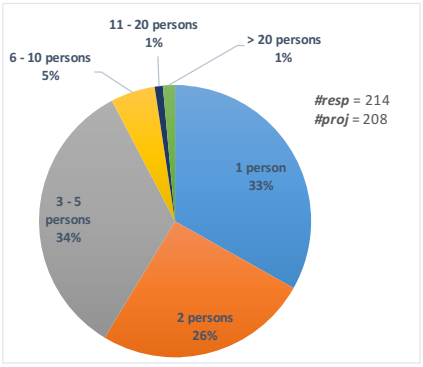


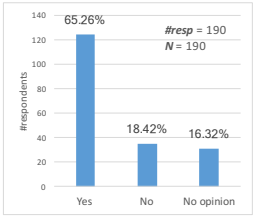
Figure 11: Number of contributors who implemented UML models in a project

Results for $SQ_{2.3}$*: Designs introduced with UML are in most cases adopted during implementation phase (fully or with slight changes). Most often these designs are implemented by groups of 2-5 persons.*

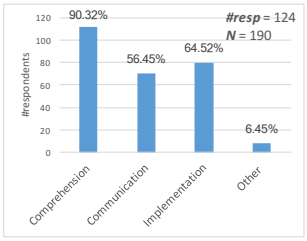
D. What is impact/benefit of UML?

1) Can UML models support new contributors?: We used two perspectives to approach the question whether UML models support new contributors.

First, we ask founders if they think that the UML models help new contributors to join their projects. We received



(a) Do UML models help new contributors?



(b) For what tasks do the models help?

Figure 12: Responses for the questions whether UML models help new contributors to join the project.

190 responses from 84 F-1UCs, 79 F-NUCs and 27 F-UCs. For those who agreed, we further asked with what tasks the models help. Fig. 12 shows the responses in detail. 124 out of 190 respondents (65.26%) agreed that UML models can help new contributors when joining the projects. They expected the models to assist new contributors in comprehending the system (90%), during implementation phases (65%), and when communicating with other contributors (56.5%).

Second, we asked each contributor what software artifacts he/she used when they got started with the project. 485 contributors answered this question. Despite the fact that most of respondents were familiar with architectural documents (as shown in Section V-A) , source code still remains their first choice to start working with an OSS project (81%) - see Fig. 13. Remarkably, UML and software models in general were reported to be starting points for 55% and 43.5% of the respondents, respectively. This is more than the proportion of contributors who started using wikis, issues, manuals, and auto-generated code documentations. This conforms the answers given by the founders about new contributors.

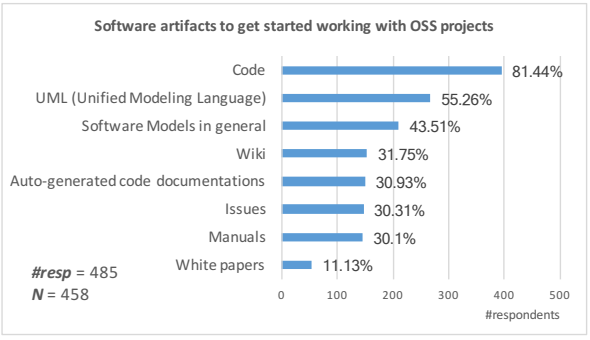
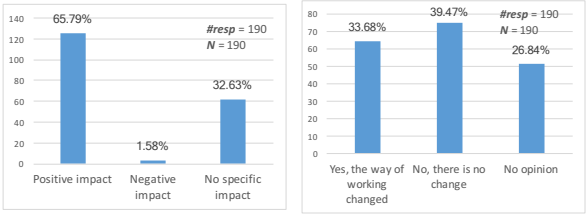


Figure 13: Software artifacts used by the respondents to get started working with their OSS project (multiple choices were allowed).

Results for $SQ_{3.1}$*: The results suggest that UML is helpful for new contributors to get up to speed.*



(a) Overall impact (b) Impacts on working routine

Figure 14: Impacts of introducing UML into OSS projects

2) What are the impacts of using UML in OSS projects?:

Because of their overview about the projects, we asked founders for their impression about the impacts of introducing UML to their project. Fig. 14a and Fig. 14b summarize the 190 answers for the two questions. A majority of respondents (65.79%) reported positive impacts, while only a few founders (<2%) encountered negative impacts. Only, 34% of the founders saw changes in the way the contributors worked after UML was introduced.

To find out more about the changes, we asked those who observed changes to describe the way the working routine had changed. We received 31 responses to the open ended question. Comments positive to UML can be summarized in following groups: i) Hiding complexity/improved overview (mentioned **18** times); ii) Improved of communication/ reduced ambiguity (6 times); iii) Prevention of sub-standard implementations (5 times); iv) Improved scoping and partitioning of work (3 times); v) Improved/easier to implement designs (9 times); vi) Improved quality assurance (1 time); vii) Reduced architecture degradation (1 time).

We also received two answers describing negative changes, complaining about more work and the need for developers to learn the UML notation.

Results for $SQ_{3.2}$*: One third of respondents reported changes of the working routine due to UML, mainly in planning phase, development process and in communication. Most reported changes can be considered positive.*

3) Can UML models help to attract new contributors?: We ask founders if they think that UML models helps to attract new contributors to join their projects. 190 founders answered this question. Fig. 12 shows the responses in detail. Only a few of the respondents (21.58%) believe that UML models can attract new contributors, while most of them think UML is not an attractive factor (47.37%).

We asked those who think UML models attract new contributors for reasons behind their thoughts. We received only 25 answers, including following arguments: a) UML models makes the project and its goals become easier to understand (mentioned 13 times), b) the potential of UML to help new contributors (by code comprehension) (7 times), c) visual documentation is considered attractive (3 times), and d) UML can support communication between old and new members (2 times).

It is worth mentioning that two of the projects have been based on executable UML diagrams (xtUML), therefore the diagrams were considered a magnet to the contributors.

However, two of the respondents put their previous answer that UML is an attracting factor in perspective, mentioning additional factors, such as the personality, the quality of the model, and complexity of the project, e.g. *"I feel that it depends on two things: how perceptive the contributors are, and how elegantly and interesting the models was structured"* .

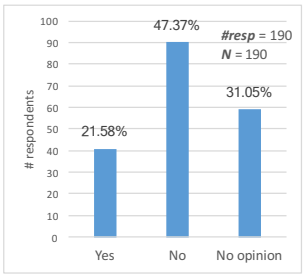


Figure 15: Do UML models attract new contributors to join the project?

Results for $SQ_{3.3}$*: Only few founders think UML models attract new contributors to join the projects.*

VI. Discussions

In the following we discuss our insights in context of related works and implications of our results. Furthermore, we discuss threats to validity.

A. Comparison to Insights to Related Works

In this section, the observations are compared with findings from related works.

Communication: First of all, the observation that UML is used for communication purposes within OSS fits to observations that were already made about the use of documentation by Kazman et al. [23] and sketches Chung et al. [24]. Furthermore, the results fit to the insight of Gorschek et al. [15], who also observed a use for communication within industrial and open source programmers.

New contributors: The observation that new contributors seem to benefit form the use of UML confirms the first anecdotal evidence that Chung et al. [24] collected. Again, Gorschek et al. [15] found similar tendencies in their survey, where the use of models was found to be higher for novices.

Design and documentation: We could uncover one main similarity in the use of UML in OSS and industry. First, we observed that UML is mainly used for design and documentation, and less for code generation within open

source. Similar observations were made for industrial usage by Torchiano et al. [14] and Forward et al. [16].

Role splits: However, we also found a hint to a contrast in the UML usage. While we observed that the architectures defined within UML models are often implemented by multiple persons, as it happens within industry, we also observed that in the most cases all of these contributors participated in the model creation. This seems to be a contrast to the practice in many industrial cases, where the people creating the models are not necessarily the persons who create the code, as, e.g., observed by Kuhn et al. [13].

Finally, we made two observations that should be further studied also in industry. *Passive benefits:* This is on the one hand the observation that many persons who do not create UML, consider its existence in the project nonetheless as beneficial. *Partial adoption:* Furthermore, we found that many models are only partially adopted during the implementation. It would be interesting to see whether this is conform or in contrast to industrial practice.

B. Implications

In most investigated aspects the answers given by NUCs showed a slight tendency to be more positive about UML than the answers of UML contributors. Thus, it seems that models have an impact on teams that affects not just the models creators positively. We hope that OSS contributors feel motivated by these results to contributing models. Furthermore, it seems that the usage of UML helps new contributors to get productive. This might be seen as an incentive for the UML introduction.

Finally, the observed contrast that most people implementing a models also participated in its creation, might be an interesting option for industrial practice, too. Especially, when agile practices are applied.

C. Threats to Validity

In the following, we discuss internal and external threats to validity of our study as introduced by Wohlin et al. [32].

Internal validity: Some threats that are generic to research that bases on GitHub, as discussed by Kalliamvakou et.al. [29], concern our study, too: First, a large amount of GitHub projects are not software development projects or have very few commits, only. Furthermore most GitHub projects are inactive (Kalliamvakou et al. guess that the amount of active projects is around 22%). To mitigate the impact of these threats on our study, we filtered the projects based on the number of commits and size. Since such filters are always just heuristics, it is probable that some of the remaining projects still are toy or educational projects. However, we consider the remaining threat acceptable, since we can assume that the vast majority of the here studied projects are real software development projects.

We focus on projects that do use UML only, to ensure that questioned developers have the experience of working in a project with UML. To ensure nonetheless that persons that prefer to not use UML are not underrepresented, we send the questionnaire not just to persons who manipulated UML, but also to contributors who did not change or introduce UML files (NUCs). Therefore, we believe that our results still provide valuable insights.

External validity: Our study focuses strongly on open source projects. While we do not expect a direct generalizability of our results to closed source projects, we expect them to be mostly generalizable to open source projects. We did not limit the domain. However, there might be a bias towards the domain that comes with the usage of UML. Since we study the impact of UML, when it is used, we consider our results valuable despite the possible bias in studies domains. Finally, we only have a look at UML models that are stored within specific formats. Of course, it would be better to have a look at all possible representations of UML models that exist. However, the selected set of formats is with the standards (.uml and .xmi) and image files already broad and allows a first valuable insight.

VII. Conclusion and Future work

In this paper we study the use of UML in open source, in order to identify commonalities and differences to the use of UML in industry. Therefore, we performed a survey with contributors from 458 GitHub projects that include UML files. Our study delivers some first insights that might help companies to decide whether to promote UML usage in open source projects. In favor of UML are the observations that UML actually helps new contributors and is generally perceived as supportive. However, UML does not seem to have the potential to attract new contributors. Further, we found that the UML use in open source projects is partially similar to the industrial use. However, there are also differences that should be considered, when joining industrial projects with open source efforts. For example, the fact that there seems to be barely a split of roles between model creator and person implementing the modeled system. Furthermore, we found that many modeled designs are only partially followed during implementation.

Future works: We only use a part of survey responses in this study (ignoring responses of short-time projects). In future, we plan to compare, whether the results for these projects are different from the ones we found. Furthermore, we plan to use meta data to investigate, whether different aspects such as size, active time, and the number of contributors of a project affect the model use and perceptions of developers within the projects.

Acknowledgment

We are very grateful to all participants of the study for taking the time and sharing their experience.

REFERENCES

[1] Cristina Gacek and Budi Arief. The many meanings of open source. *IEEE software*, 21(1):34–40, 2004.

[2] Kevin Carillo and Chitu Okoli. The open source movement: a revolution in software development. *Journal of Computer Information Systems*, 49(2):1–9, 2008.

[3] Brian Fitzgerald. The transformation of open source software. *Mis Quarterly*, pages 587–598, 2006.

[4] Daniel M German. The gnome project: a case study of open source, global software development. *Software Process: Improvement and Practice*, 8(4):201–215, 2003.

[5] Dirk Riehle. The economic case for open source foundations. *Computer*, 43(1):0086–90, 2010.

[6] Øyvind Hauge, Claudia Ayala, and Reidar Conradi. Adoption of open source software in software-intensive organizations–a systematic literature review. *Information and Software Technology*, 52(11):1133–1154, 2010.

[7] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys (CSUR)*, 44(2):7, 2012.

[8] Klaas-Jan Stol, Muhammad Ali Babar, Paris Avgeriou, and Brian Fitzgerald. A comparative study of challenges in integrating open source software and inner source software. *Information and Software Technology*, 53(12):1319–1336, 2011.

[9] Diomidis Spinellis and Clemens Szyperski. How is open source affecting software development? *IEEE Software*, 21(1):28, 2004.

[10] Claudia Hauff and Georgios Gousios. Matching github developer profiles to job advertisements. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 362–366. IEEE Press, 2015.

[11] Regina Hebig, Truong Ho-Quang, Gregorio Robles, Michel R.V. Chaudron, and Miguel Angel Fernandez. The quest for open source projects that use uml: Mining github. *International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, 2016.

[12] 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.

[13] Adrian Kuhn, Gail C Murphy, and C Albert Thompson. An exploratory study of forces and frictions affecting large-scale model-driven development. In *International Conference on Model Driven Engineering Languages and Systems*, pages 352–367. Springer, 2012.

[14] Marco Torchiano, Federico Tomassetti, Filippo Ricca, Alessandro Tiso, and Gianna Reggio. Relevance, benefits, and problems of software modelling and model driven techniques - A survey in the italian industry. *Journal of Systems and Software*, 86(8):2110–2126, 2013.

[15] Tony Gorschek, Ewan Tempero, and Lefteris Angelis. On the use of software design models in software development practice: An empirical investigation. *Journal of Systems and Software*, 95:176–193, 2014.

[16] Andrew Forward, Omar Badreddin, and Timothy C Lethbridge. Perceptions of software modeling: a survey of software practitioners. In *5th workshop from code centric to model centric: evaluating the effectiveness of MDD*, 2010.

[17] Paul Baker, Shiou Loh, and Frank Weil. Model-driven engineering in a large industrial contextmotorola case study. In *International Conference on Model Driven Engineering Languages and Systems*, pages 476–491. Springer, 2005.

[18] Ariadi Nugroho and Michel RV Chaudron. Evaluating the impact of uml modeling on software quality: An industrial case study. In *International Conference on Model Driven Engineering Languages and Systems*, pages 181–195. 2009.

[19] Omar Badreddin, Timothy C. Lethbridge, and Maged Elassar. *Modeling Practices in Open Source Software*, pages 127–139. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[20] Wei Ding, Peng Liang, Anthony Tang, Hans Van Vliet, and Mojtaba Shahin. How do open source communities document software architecture: An exploratory survey. In *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*, pages 136–145. IEEE, 2014.

[21] Koji Yatani, Eunyoung Chung, Carlos Jensen, and Khai N Truong. Understanding how and why open source contributors use diagrams in the development of ubuntu. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 995–1004. ACM, 2009.

[22] Mohd Hafeez Osman and Michel R. V. Chaudron. UML usage in open source software development : A field study. In *Proceedings of the 3rd International Workshop on Experiences and Empirical Studies in Software Modeling co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, pages 23–32, 2013.

[23] R. Kazman, D. Goldenson, I. Monarch, W. Nichols, and G. Valetto. Evaluating the effects of architectural documentation: A case study of a large scale open source project. *IEEE Transactions on Software Engineering*, 42(3):220–260, 2016.

[24] Eunyoung Chung, Carlos Jensen, Koji Yatani, Victor Kuechler, and Khai N Truong. Sketching and drawing in the design of open source software. In *Visual Languages and Human-Centric Computing (VL/HCC), 2010 IEEE Symposium on*, pages 195–202. IEEE, 2010.

[25] Philip Langer, Tanja Mayerhofer, Manuel Wimmer, and Gerti Kappel. On the usage of uml: Initial results of analyzing open uml models. In *Modellierung*, volume 19, page 21, 2014.

[26] Georgios Gousios and Diomidis Spinellis. Ghtorrent: Github's data from a firehose. In *Mining software repositories (msr), 2012 9th IEEE working conference on*, pages 12–21. 2012.

[27] Hudson Borges, André C. Hora, and Marco Tulio Valente. Understanding the factors that impact the popularity of github repositories. *CoRR*, abs/1606.04984, 2016.

[28] 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 (IJOSSP)*, 1(1):24–45, 2009.

[29] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 92–101, New York, NY, USA, 2014. ACM.

[30] Igor Scaliante Wiese, Igor Steinmacher, Christoph Treude, Jose Teodoro Da Silva, and Marco Gerosa. Who is who in the mailing list? comparing six disambiguation heuristics to identify multiple addresses of a participant. In *Proceedings of the 32nd International Conference on Software Maintenance and Evolution*, 2016.

[31] Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, 25(4):557–572, 1999.

[32] Claes Wohlin, Per Runeson, Martin Höst, Magnus Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.