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

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).

[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.

[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

978-1-5386-1544-7/17 $31.00 © 2017 IEEE
DOI 10.1109/MSR.2017.48

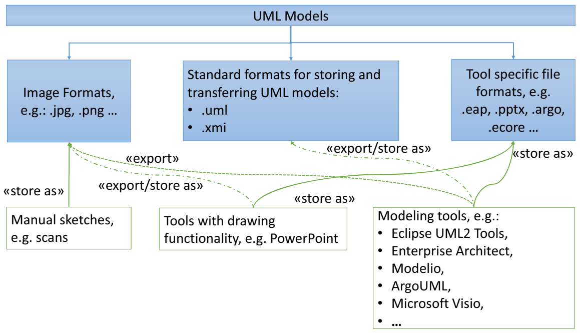
UML Models
Image Formats, e.g.: .jpg, .png ...
Standard formats for storing and transferring UML models:
• .uml
• .xmi
Tool specific file formats, e.g. .eap, .pptx, .argo, .ecore ...
«export/store as»
«store as»
«export»
«export/store as»
«store as»
«store as»
Manual sketches, e.g. scans
Tools with drawing functionality, e.g. PowerPoint
Modeling tools, e.g.:
• Eclipse UML2 Tools,
• Enterprise Architect,
• Modelio,
• ArgoUML,
• Microsoft Visio,
• ...

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.

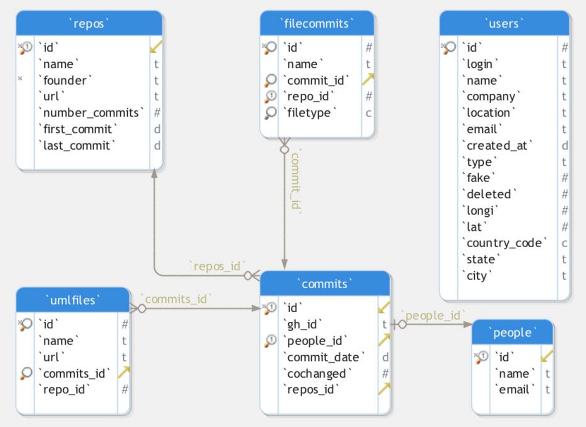

Fig. 2: The relational database schema

- **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].

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.