

UML is Back. Or is it?

Investigating the Past, Present, and Future of UML in Open Source Software

Joseph Romeo, Marco Raglianti, Csaba Nagy, Michele Lanza
REVEAL @ Software Institute – USI, Lugano, Switzerland

Abstract—Since its inception, UML, the Unified Modeling Language, has been touted as the way to go when it comes to designing and documenting software systems. While being an integral part of many university software engineering programs, UML has found little consideration among developers, especially in open source software. Reasons for this include that UML shares some shortcomings with other forms of documentation (*e.g.*, limited availability, outdatedness, inadequate level of detail).

We present a study to investigate the evolution and the current situation regarding the use of UML in open source projects. We mined and analyzed ~13k GitHub projects, developing strategies and heuristics to identify UML files through their extensions and contents, for a quantitative analysis of two decades of evolution of the usage of UML. We explored the popularity of UML, derived characteristics of projects leveraging UML, and analyzed the authors, creators and maintainers, of UML artifacts.

Our study confirms that UML is indeed still under-utilized. At the same time we found evidence of a resurgence coinciding with the popularity of human-readable text-based formats, defined and used by tools like PlantUML and Mermaid. We discuss how identifying and addressing the new challenges implied by this resurgence could impact the future of UML.

Index Terms—UML, human-readable text-based UML, software design evolution, software documentation evolution

I. INTRODUCTION

Andrew Watson said that the history of visual modeling can be divided cleanly into two eras, “Before UML” and “After UML” [1]. The period before UML was a time of division and strife, named the “Method Wars” era [2]. Booch, Rumbaugh, and Jacobson, a.k.a. the “Three Amigos”, creators of three popular object-oriented development approaches [3]–[5], combined their efforts towards a single, Unified Modeling Language (UML) [6]. The adoption of UML as a standard by the Object Management Group in 1997 marked the end of the method wars and the beginning of the “After UML” era.

UML is one of the most prominent modeling languages used in software design and development [7]. It has been used for automatic code generation for embedded systems [8], to create workflow diagrams for business processes [9], [10], to design [7], measure [11], maintain [12], comprehend [13], and reuse [14], [15] software systems. We focus on UML usage in *designing and documenting* software systems.

Despite the apparent popularity of UML in academic publications, its history hints at a progressive stagnation. Minor revisions of the UML specification have been issued with an average yearly pace between 1997 and 2003 [16]. UML 2.0 took over two years to be released, losing the momentum.

Last but not least, UML 2.5.1, the latest version of the specification, dates back to December 2017 without any updates in more than six years.

One of the main shortcomings of UML from a practitioner’s perspective is the availability and quality of its tool support [17]. UML’s decline in popularity since 2004¹ exacerbated the problem. This decline culminated with the discontinuation of language support in one of the major Integrated Development Environments (IDEs): Microsoft Visual Studio removed UML support in 2016 due to underutilization by clients, to focus development efforts on features deemed more relevant for a larger share of the user base.²

Given the benefits of UML diagrams for system comprehension, design, and documentation (*e.g.*, UML studies with the Lindholmen dataset [18]), we aim to understand why UML is still underutilized, focusing on Open Source Software (OSS).

We investigated ~13k OSS repositories to find those containing UML, identifying commonalities and differences. After tagging all UML files through their extensions and contents, we found that only 4.2% of the repositories in our dataset (552 out of 13,152) contained UML diagram files in the last 20 years. In the same time-span, UML file formats have significantly changed. Our findings show the progressive abandonment of dedicated UML graphical tools first, and changes in the role of IDE plugins later.

More importantly, we found empirical evidence of a resurgence of UML, after 2017, coinciding with the advent of human-readable, versionable, and easily maintainable text-based UML files (*e.g.*, PlantUML, Mermaid). This resurgence did not modify the natural tendency to neglect documentation and its maintenance. We conclude by analyzing the relationships among UML, repository activity, community, and human factors, focusing on UML artifact authors and their commits.

UML’s decline in popularity and language stagnation are now countered by a renewed interest in human-readable text-based UML. Dropping special tools in favor of generic text editors, these formats introduce new challenges. For example, each tool uses a slightly different subset of UML, sometimes with an ambiguous specification. Moreover, an effective and consistent layout of visual elements is usually impossible with these tools, severely impacting their effectiveness [19].

¹See <https://tinyurl.com/google-trends-uml>

²See <https://www.infoworld.com/article/3131600/>

We argue that Watson’s split needs to be revised and that a third era has already started. Only a convergence between language specification evolution, UML human-readable text file formats standardization, and improved tool support can address these issues for a (new or real?) golden age of UML.

II. UML AND THE STATE OF THE ART

UML is a visual modeling language with a formal specification, standardized in 1997 by the Object Modeling Group. Touted as the de facto standard for modeling software systems, UML allows developers to visually describe the system, what it does, and how. Specialized software is used to create UML diagrams [20], with an emphasis on their syntactic correctness.

Software engineering courses in university computer science curricula teach UML to students as a tool to reason about software, its design, and its documentation [21]. The relevance of UML in teaching is not matched by its usage in practice, though: In a survey with 80 software architects, Lange *et al.* found that UML in industry only loosely adheres to the official specification [22]. Nevertheless, UML is a valid instrument to teach topics related to software architecture. Rukmono and Chaudron found that peer feedback on UML diagrams can improve students’ learning of software design [19].

High-quality documentation increases the success chances of a software project [23]–[25]. This is especially true for UML diagrams and documentation based on visual modeling. Software design diagrams promote active discussion in homogeneous and cross-functional teams [26]. Developers achieve better functional correctness when making changes with accurate and up-to-date design diagrams [23], [27].

Although most results agree on the usefulness of UML, some exceptions exist. Scanniello *et al.* found that UML models produced in the requirements analysis process influence neither the comprehensibility nor the modifiability of source code [28]. Gravino *et al.* found that the time to accomplish comprehension tasks for less experienced participants is negatively affected when UML design models are provided in addition to source code [29]. There is a necessary trade-off between correctness, quality, and time to complete software maintenance tasks when leveraging UML diagrams [23].

UML drawing software tools leverage the visual nature of the language, providing drawing capabilities similar to whiteboard sketches, mixing pen and paper with digital means [30]–[32]. Bergström *et al.* investigated the automated assessment of UML class diagram layouts via machine learning, since element spacing and rectangle orthogonality play a fundamental role in the perceived quality of a UML class diagram [33]. Human-readable text-based UML takes away control from the diagram’s creator, for example, on how elements are laid out, their spacing, and where connecting lines are drawn. This impairs what Rukmono and Chaudron found to be key aspects of UML diagrams’ understanding and usability [19].

Badreddin *et al.* investigated the underutilization of models in OSS and a potential language-based solution: Umple [34]. Robles *et al.* and Hebig *et al.* already investigated UML models in the lifespan of GitHub projects by focusing on

images and a limited subset of non-image file extensions (.uml and .xmi) [35], [36]. While the classifier proposed by Ho-Quang *et al.* [37] has been leveraged to identify class diagrams, manual analysis was performed on the remaining images to discriminate other types of diagrams. Our work focuses on the evolution of text-based representations of UML as a more reliably parsable and easily manageable format and the Umple format is one of those we analyzed.

Previous studies used surveys with developers and software practitioners to investigate their perception and usage of software modeling [38]. Ho-Quang *et al.* analyzed the role of modeling in OSS development [39]. Analyzing projects containing UML files, they used a method similar to ours to retrieve UML files for project selection, but our study aims to address different research questions and to provide empirical evidence based on the analysis of a large dataset of UML artifacts spanning 20 years. Besides updating the analysis of UML usage in OSS with the trends in the last 6 years, the phenomenon emerging between 2017 and 2022 present new and distinctive characteristics that warrant specific attention.

The resurgence of UML with human-readable text-based formats, one of the main findings of the present study, is indeed an emerging phenomenon that prompts for better approaches and tools. For example, we need to address layout issues, without giving back the ease of use and maintainability gained by making the graphical representation almost a side-effect.

III. RESEARCH QUESTIONS

Given the advantages that the use of UML should provide, according to the literature, and the fact that a great effort is spent in teaching the language in universities, the fundamental meta-question of our analysis is:

META-QUESTION

Why is UML still underutilized?

As any why question has no imminent answer, we focus in the following on four distinct research questions pertaining to the format, diffusion, demographic, and the designers of UML diagrams, which we answer through an empirical study.

We define a strategy to semi-automatically capture UML artifacts, based on file formats. This necessary step to collect data on the use of UML leads to our first research question:

RQ₁ – FORMAT

What formats are UML diagrams found in?

With a dataset of 20 years of history of semi-automatically gathered UML artifacts, we focus on the evolution of UML usage trends at large, from the beginning to the current state:

RQ₂ – DIFFUSION

How widespread is UML in open source software?

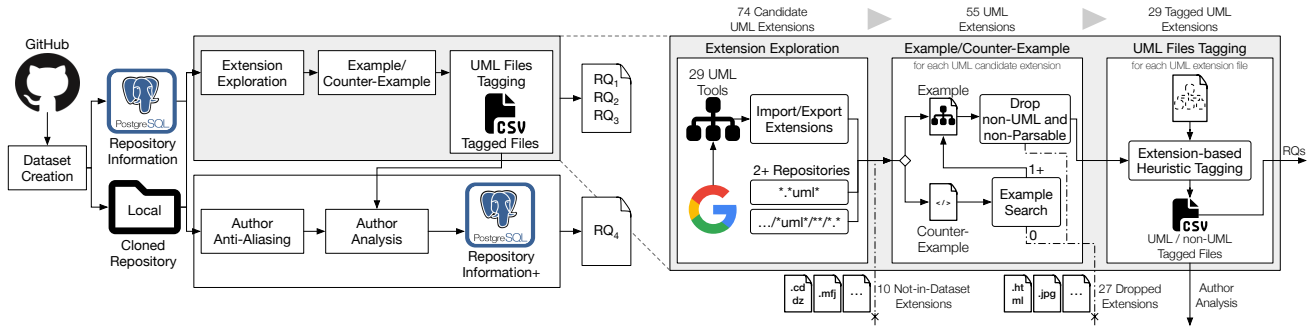


Fig. 1. Approach Overview (left) and Detailed UML Extraction (right)

Different characteristics of the projects using UML, with respect to activity (*e.g.*, commits, releases), participation (*e.g.*, pull requests, contributors), or community engagement (*e.g.*, watchers, stars), could be used to advocate for the ubiquitous adoption of UML. If UML would contribute positively to certain metrics (*e.g.*, participation), its use should be further encouraged based on these benefits. Otherwise, the meta-question might be dismissed by just saying that UML is under-utilized because its practical usefulness is overestimated. Which leads to:

RQ₃ — DEMOGRAPHIC

What types of projects include UML diagrams?

Finally, our focus shifts from repositories to contributors to analyze the characteristics of key figures in the lifecycle of UML artifacts. If no significant difference can be found in the repositories, can human factors related to UML contributors (those actively working with UML artifacts) motivate the under-utilization of UML? Hence our final research question:

RQ₄ — DESIGNERS

Who is creating and maintaining UML diagrams?

IV. DATASET CREATION

We used the SEART GitHub Search³ (GHS) tool from Dabic *et al.* [40] to gather an initial set of GitHub projects. After excluding forks, to avoid “polluting” the dataset with social forks [41], [42], we selected projects with at least 2,000 commits to eliminate toy repositories. We selected projects with at least 10 contributors to ensure the need for collaboration, which increases the utility of diagrams. We selected projects with at least 100 stars to ensure projects are considered useful by the OSS community.

Our final dataset consists of 13,152 repositories, which we cloned locally on April 1, 2023. We performed a deep cloning for each project to analyze their entire history. We extracted with *cloc* [43] the number of lines of code (LOCs) for each project and removed those with less than 10k LOCs.

³See <https://seart-ghs.si.usi.ch/>

cloc counts LOC in many different languages, we found 278 of them, for example C, C++, but also C/C++ Headers and JSON. We used *cloc* mainly to remove from our dataset emptied projects which kept the statistics we used as GHS filtering criteria. We stored summary information about each repository in a PostgreSQL database. Table I provides descriptive statistics about the complete dataset.

TABLE I
STATISTICS OF THE DATASET WITH 13,152 PROJECTS

Metric	Min	Median	Mean	Max	Total
Commits	2,000	4,214	9,506	841 K	125.0M
Contributors	10	63	112	13 K	*1.5M
Files	7	849	2,330	128 K	30.6M
LOCs	10,014	127,785	485,026	46M	6.4 B
Stars	100	708	3,201	322 K	42.1M

*Non-unique total contributors.

V. MINING AND ANALYZING UML REPOSITORIES

In Figure 1, we show an overview of our approach to mining and analyzing the repositories on which we base our study.

We split the meta-question into four research questions (RQ_{1–4}) on the usage and the nature of UML in OSS repositories. For each research question we give an overview, then we present our methodology, results, and findings.

A. RQ₁: What Formats are UML Diagrams Found in?

File extensions are an efficient way to discriminate file types, but they are not sufficient to uniquely identify UML diagrams. Thus, to obtain a “handle” on UML artifacts useful to address all the RQs, we implemented strategies to manually find examples and counter-examples of UML diagrams for different file extensions. For each potential UML file extension (*i.e.*, those with at least a UML example), we devised heuristics to tag semi-automatically the files that contain UML. In the following we describe each step of the UML extraction procedure. In Figure 1 (right), we show a “zoomed-in” overview.

Extension Exploration. We created a list of candidate UML extensions using the following approach: We searched for “top UML diagramming tools” on Google and identified 29 popular tools for creating UML diagrams. We downloaded and installed each tool to determine the supported file extensions. We checked their import, export, and save functionalities.

We could not install or run 5 programs, for example, due to licensing issues. For those, we relied on the documentation or YouTube videos to determine which file extensions were supported. We complemented this approach by searching for all extensions with “uml” in the name or in any part of the file path and present in at least two repositories. After generating the list of candidates, we removed any extensions for which we found no examples in our dataset (*i.e.*, a tool’s unused import/export format), along with any extension used by known non-UML file types (*e.g.*, .java, .jar, .am).

Example and Counter-Example. For each file extension we searched for an example and a counter-example. Extensions for which we could find at least an example are valid UML extensions. Those for which we could find at least one counter-example are not UML-specific. We randomly selected, retrieved, and manually inspected one file for each extension, to confirm whether it was a UML diagram or not.

We were left either with an example or a counter-example. If, for example, we had a counter-example for .txt files, we then searched all .txt files to find an example of UML. In this search we tried to reduce the number of files to be inspected before finding an example. If we were looking for a UML example, we first analyzed the files whose path contains the string “/uml/”. Then we searched for the following keywords in the file names (case insensitive): *architecture, uml, diagram,*

sequence, class, usecase, state, activity, component, deployment, object, communication, composite, interaction, collaboration, package, profile, and timing (part of this keyword list is comprised of the names of the 14 UML diagram types). We inspected these files first. For extensions with more than 150 files, we explored a statistically significant sample (confidence level = 95%, margin of error = 5%). Then, we removed extensions that are too generic (*e.g.*, .html, .md)⁴ and formats whose content we cannot parse automatically (*e.g.*, .jpg, .png).

To recap, the procedure has multiple steps with branches (see Figure 1 right). If a randomly sampled file for an extension is an example (UML file with that extension), we mark the extension as UML (not yet exclusive). Then, we look for counter-examples for the extension to see if this is a UML-exclusive extension or if we can find both UML and non-UML files with this extension. If we find a counter-example, this is not a UML-exclusive extension. If the first randomly selected file is a counter-example (not a UML file), with no examples found in the second step, it corresponds to a format to discard (no UML files found with that extension).

⁴The exclusion of these file extensions is a potential limitation of our study but it is mitigated by the fact that, to the best of our knowledge, no tools except EnterpriseArchitect can import HTML or Markdown files to produce UML diagrams, rendering the collaborative maintenance of such artifacts impractical (see `uml-tools-extensions.md` in the replication package for a list of extensions importable and exportable by each tool).

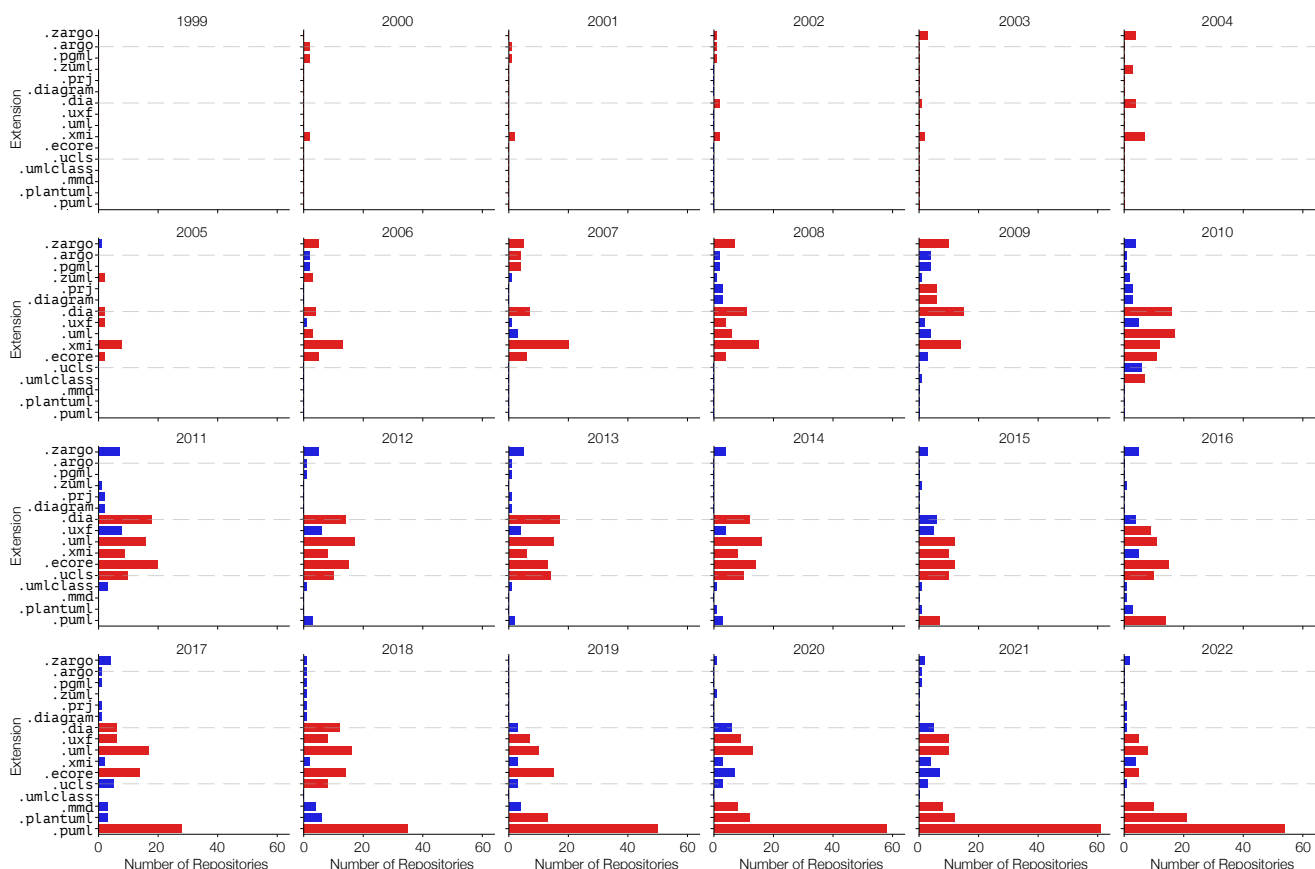


Fig. 2. Number of Repositories For Most Popular UML Files Modified Each Year. Top 5 Extensions for Each Year in Red.

UML Files Tagging. Some file extensions are not used for UML purposes exclusively (e.g., .dia, .drawio, .graffle). We tag each UML file in our dataset not only based on its extension, but by semi-automatically parsing its content.

For each file with an extension coming from the previous step, we detect UML content via extension-specific regular expressions (after de-compressing files). For example, .ecore files are UML files for the Eclipse Modeling Framework if they contain either the word “EClass” or “EPackage”. By devising such strategies we automatically tagged almost every UML file in the dataset. Where an automatic approach was not feasible, we checked the files manually. This was needed for 26 files of 3 extensions (.asta, .cmof, .yuml). This step consisted in a progressive refinement of the strategies until all files with all extensions resulted in a UML or non-UML tag. Table II shows the UML extensions considered in the rest of the study.

TABLE II
REPOSITORIES WITH TAGGED UML FILES
(OUT OF 552 REPOSITORIES WITH UML, SEE SECTION V-B)

Extension	# Repos	Extension	# Repos
.puml	160	.diagram	8
.uml	100	.pgml	8
.dia	67	.prj	8
.xmi	60	.zuml	7
.plantuml	48	.asta	6
.ecore	43	.session	6
.ucls	33	.iuml	5
.zargo	32	.mdzip	5
.uxf	30	.yuml	5
.mmd	26	.gliffy	3
.mdj	18	.platuml	2
.vpp	14	.umlprofile	2
.pu	10	.cmof	1
.umlclass	9	.ump	1
.argo	8		

There are two assumptions we made: (i) for each file we examine only its first version, assuming that the nature of a file does not change, only its content does; (ii) if multiple files with the same name and extension exist in different paths, we analyze only one of them, assuming that files with the same name have the same type. This simplification for the manual analysis impacts only identification of the tagging strategies. The applied heuristics tag each file in the different paths as UML or non-UML and each file is counted independently.

Results. Figure 2 shows the evolution of the most popular UML extensions by number of repositories using them, since 1999. For each year, we plot the number of repositories actively making changes (with at least one commit) to UML files for each extension. The red bars represent the top 5 extensions found in the most repositories for that year.

The .xmi, .argo, .pgml, .zargo, and .dia are the only UML extensions used from 2000 to 2003. The .zargo, .argo, and .pgml extensions are used by ArgoUML, and .xmi is a standard for many graphical UML tools.⁵ Dia is a diagramming tool.

⁵It is worth noting again in this context that extensions such as .xmi can also be used for non-UML content. As described previously in this section, we consider only the UML-tagged files for each extension. The regular expressions we devised count only .xmi files with actual UML content. The same applies to other generic extensions (e.g., .dia, .ecore).

2004 to 2007 is still dominated by .xmi and .zargo, but .dia’s use increases and .ecore appears. The Eclipse Modeling Framework (EMF), an Eclipse plugin for graphical modeling and model-based automatic code generation, uses .ecore files.

From 2008 to 2015, .xmi, .dia, and .ecore remain popular while .ucls, .uxf, and .uml start to become popular. The .uml extension contains many types of UML files, including Ecore/EMF, XMI, and PlantUML files. The .ucls extension is used by the ObjectAid UML Explorer Eclipse plugin. The .uxf extension is the UML eXchange Format (similar to XMI).

From 2016 to 2022, the previously popular extensions are supplanted by .mmd (Mermaid), .plantuml, and .puml (PlantUML). In 2022, the two text-based UML diagramming tools (i.e., PlantUML and Mermaid) are the most used.

Given the above, we can see three main periods. Early on, from 2000 to 2007, we see the usage of graphical UML modeling tools and generic diagramming tools. From 2008 to 2016, Eclipse plugins took over as the most popular. Finally, from 2016 to 2022, we see the rise of text-based UML.

Interestingly, PlantUML’s use did not rise until 2016, despite being released in 2009. Similarly, Mermaid released in 2014, gained popularity in 2016, and made it into the top 5 extensions in 2020. One event that possibly contributed to this shift is the release of Visual Studio Code (VSCode) in 2015.

Given how popular Eclipse plugins were from 2008 to 2016, developers liked UML modeling tools to be integrated into their IDE. Since its release in 2016, VSCode has become the most popular IDE for developers.⁶ It has extensions (i.e., plugins) for UML diagrams in PlantUML, Mermaid, and UMLet.

From 2016 to 2022, we see a huge uptick in the overall number of repositories actively working with UML, mainly due to the rise of PlantUML. This could be due to the advantages that text-based UML modeling tools offer. Compared to graphical UML modeling tools, it is arguably easier to version control text-based UML diagrams, for example, having meaningful *diffs* that can be checked for correctness during code reviews.

Summarizing: RQ₁ investigates how UML tools and formats evolved in the last two decades. We present the trends, highlighting the novel ones and the recent history. The *underutilization* of UML emerges from the number of repositories containing UML artifacts. The *Why* of the meta-question is related to the tools and formats themselves. For example, a fragmented ecosystem of tools has maintainability and interoperability problems, leading to underutilization (Section VI).

RQ₁ – FINDINGS

- F₁ PlantUML has been the most popular UML diagramming tool since 2016, UML’s resurgence year.
- F₂ Mermaid (i.e., .mmd) surpassed .uxf and .uml extensions in 2022. The other human-readable text format becomes the second most popular after PlantUML.
- F₃ Text-based UML diagramming, over the past few years, has supplanted UML graphical editors.

⁶See <https://survey.stackoverflow.co/2022>

B. RQ₂: How Widespread is UML in Open Source Software?

For this RQ, we analyze the history of UML’s diffusion to (re-)assess the current status of the practice, providing insights on the evolution that brought UML to be an underutilized language. We look at the percentage of repositories that contain at least one UML diagram since their creation. We analyze the popularity and activity on UML files.

Method. After identifying candidate UML extensions and tagging UML files for RQ₁, we further restrict the dataset we first introduced in Section IV to repositories containing UML diagrams, resulting in 552 repositories. Table III summarizes descriptive statistics about the dataset of UML repositories.

TABLE III
STATISTICS OF THE DATASET WITH 552 UML REPOSITORIES

Metric	Min	Median	Mean	Max	Total
Commits	2,001	6,866	14,333	157 K	7.9M
Contributors	10	62	102	441	*56.3K
Files	60	2,038	4,254	60 K	2.3M
LOCs	10,179	302,268	838,141	13M	462.7M
Stars	100	533	2,427	80 K	1.3M

*Non-unique total contributors.

Results. The usage of UML in our dataset is not widespread. We found only 4.2% of repositories (552 out of 13,152) that contained at least one UML diagram at some point in time.

We investigated whether the usage of UML diagrams has increased or decreased over time. Since the number of active repositories changes over time, we normalize the number of repositories with UML diagrams over the number of active repositories per year (*i.e.*, having at least one commit on the main branch in that year).

Figure 3 shows the percentages of repositories with UML diagrams and of repositories actively modifying UML files (at least one UML file commit in the year) since the year 2000.

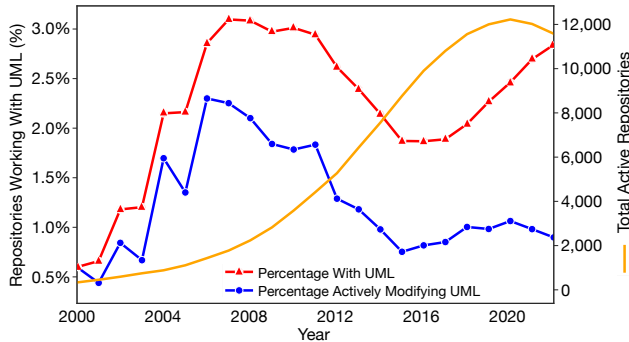


Fig. 3. Percentage of Repositories With UML

At its peak, about 3.0% of active repositories contained a UML diagram. The fraction of repositories actively modifying UML is even lower (*i.e.*, at a maximum of 2.3% of repositories modifying at least one UML file in 2006). The current percentage of repositories actively using UML (blue, circles) hovers around 1.0%, slightly above the latest local minimum in 2015. Less than one third of the projects containing UML saw activity in UML files in 2022.

The overall percentage of repositories with UML (red, triangles) follows a similar trend before 2015, but it has been increasing since 2017. The use of UML peaks around 2007, but in the last five years the divergence between projects merely containing UML and those actively modifying it has been increasing.

Summarizing: RQ₂ takes a step back and considers aggregate UML usage and maintenance data. We imply, as further discussed in Section VI, that if we do not take into account the shortcomings of the new wave of UML tools and formats, this resurgence wave might not bring UML out of the underutilization condition (*i.e.*, Why UML will remain underutilized).

RQ₂ — FINDINGS

- F₁ UML in OSS is not widespread, with a peak of only 3.0% of active repositories having a UML diagram.
- F₂ UML hit peak diffusion in 2007, then it decreased to below 2.0%, just to see a steady resurgence that is still going on since 2017.
- F₃ Usage and active maintenance of UML follow different trends since 2016. The first is increasing while the second stays almost stable in percentage.

C. RQ₃: What Types of Projects Include UML Diagrams?

Assuming that UML’s presence should influence or be influenced by the development process, we hypothesize that a measurable effect should appear in some metrics related to the repository activity or to the community of developers contributing to the project. To confirm or refute this hypothesis, we analyzed what types of projects contain UML diagrams, comparing projects by their main language and 10 GitHub metrics related to activity and community (*e.g.*, commits, issues, contributors).

Method. To analyze correlations between project metrics and the presence of UML artifacts, we compared UML and non-UML repositories with respect to the following metrics: *commits*, *contributors*, *forks*, *open issues*, *open pull requests*, *releases*, *stars*, *total issues*, *total pull requests*, and *watchers*.

For each metric, we performed a D’Agostino-Pearson normality test to verify if the data was normally distributed. Since in both cases the data was not normally distributed, we used the Mann-Whitney U test on each metric to determine if there was any statistically significant difference. For those metrics where we found a statistically significant difference between the two samples, we used Cliff’s delta to determine the magnitude of the difference.

We were also interested in possible correlations between the main language of the repository and the usage of UML. Therefore, we grouped UML repositories by main language, as reported by GitHub’s language statistics.

Results. Given that UML diagrams are used to describe various aspects of a system, we hypothesized that repositories with higher activity and larger communities would be more likely to have UML diagrams. This was not the case.

The two groups, of UML and non-UML repositories, have a statistically significant difference on the number of commits, open and total pull requests, stars, and open issues (Mann-Whitney U test $p \leq 0.01$), but only the number of commits has a non-negligible effect (positive correlation with presence of UML artifacts, Cliff's delta $\delta = 0.30$, small effect).

We investigated the potential effect of the choice of programming language on the use of UML. In Figure 4, we show the percentage of repositories with UML by main programming language.

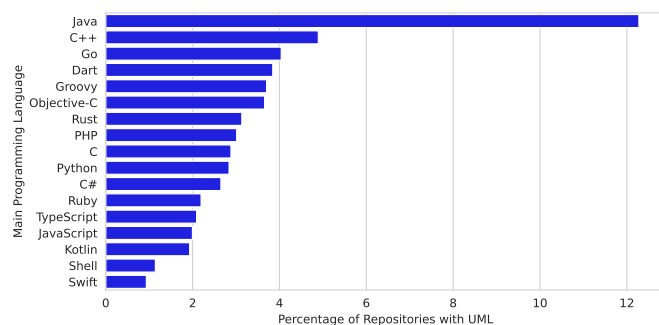


Fig. 4. Percentage of Repositories with UML by Language

We see Java as absolute first by a large margin, followed by C++ and the others with less than half its popularity. The choice of programming language seems to affect how likely a repository is to have UML. Java repositories are three times more likely to have UML than repositories in other languages.

The lack of a clear separation between languages, except for Java, makes it difficult to identify common distinguishing characteristics among those more likely to have UML diagrams. Java and C++, which support class-based object-oriented programming (OOP), are in the top spots. Following them, at number three, there is Go, which does not have classes. Then, again, we find languages which support class-based OOP near the bottom of the list, like Kotlin and Swift. Since UML was built to support OOP methodologies, it is also interesting to find C, which is not an OOP language, above Python, C#, and Kotlin.

Although we see a clear difference in the likelihood of a repository to have UML depending on its main language, we do not see any patterns in the types of languages for which this is more likely. As part of our future work, we plan to further investigate the relationships between all the project's languages (*i.e.*, not only the main one) and the use of UML.

Summarizing: RQ₃ investigates if the *Why* of the meta-question is connected to any significant improvement in relevant metrics. Apart from a slightly increased number of commits, the activity and community metrics we examined were not affected by the presence of UML artifacts in the repository, an argument in favor of the underutilization (*i.e.*, UML is underutilized because it does not improve developers' engagement with the project).

RQ₃ – FINDINGS

- F₁ Repositories with UML have a slightly but significantly higher number of commits.
- F₂ Except for the number of commits, there is no difference in activity and community metrics between UML and non-UML repositories.
- F₃ Java is the most likely programming language to use UML, with 12% of repositories (4 times the average of the other languages).

D. RQ₄: Who is Creating and Maintaining UML Diagrams?

The use of UML diagrams does not seem to influence the metrics of activity in the repository we considered, but interesting relationships could be revealed considering the project contributors. This question needs a thorough analysis of proxies such as the contribution period (*i.e.*, as a proxy for involvement in the project).

We analyze creators and maintainers of UML diagrams compared to developers. We investigate how these roles overlap and the characteristics of authors with dedicated project management roles contributing to UML diagrams.

Method. To answer RQ₄, we needed to integrate authors' information with the list of tagged files, after removing author aliases [44]–[47]. We expanded on the work of Gote and Zingg [47] by tuning the algorithm to GitHub repositories. We eliminated names and emails that we found were common placeholders in our dataset (*e.g.*, anon@github.com). We specifically ignored matches of the following names: *unknown*, *anonymous*, *anon*, and *none*. We also excluded emails containing the words *unknown*, *anonymous*, *devnull*, *noreply*, *none@none*, and *root@localhost*. In addition, we considered names in the domain of email addresses for those who host their own email accounts (*i.e.*, mail@jdoe.com). When extracting the email, we compare the domain instead of the base if the first or last name appears in the domain. The PostgreSQL database is augmented with the resulting derivative information (*e.g.*, author anti-aliasing, author analysis). A complete dump of the final database is available for replication and further exploration on [figshare](https://doi.org/10.6084/m9.figshare.28008434).⁷

Contribution Period of UML vs. non-UML Committers.

The first attribute we analyze is the average contribution period: The period between the first and the last commit of a contributor. In Figure 5, we show three box plots with the distribution of the average contribution period of UML and non-UML committers. The top box plot shows the distribution without filtering, and the middle and the bottom ones are obtained by removing contributors with less than 2 and 10 commits, respectively. The box plots of UML committers are only slightly affected by filtering. On the contrary, the median contribution period of the non-UML committer box plots moves from 0 days with no filtering to 801 days when filtering out contributors with less than 10 commits.

⁷See <https://doi.org/10.6084/m9.figshare.28008434>

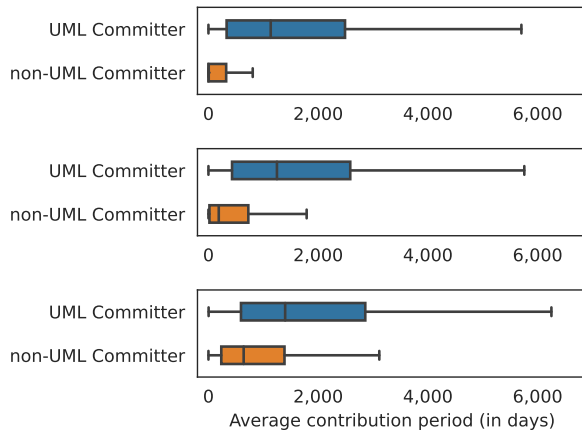


Fig. 5. Average Contribution Periods of UML vs non-UML Committers. All Contributors (top), Contributors With at Least 2 Commits (middle), and Contributors With at Least 10 Commits (bottom).

This huge shift is due to the large number of occasional contributors in terms of source code. This phenomenon is almost non-existent for UML committers. For this reason, in the following analyses, we show only the results after removing contributors with less than 10 commits.

The median contribution period of UML committers is 1,735 days versus 801 days for non-UML committers. We confirmed that this difference is statistically significant (Mann-Whitney U test $p \leq 0.01$, Cliff's delta $\delta = 0.56$, large effect). UML committers contribute to the repository for a significantly longer period (more than twice).

In Figure 6, we see a scatter plot of the same average contribution where each point represents a repository. Any point above the red line is a repository where the average contribution period of the authors of UML commits is greater than the average contribution period of the other authors.

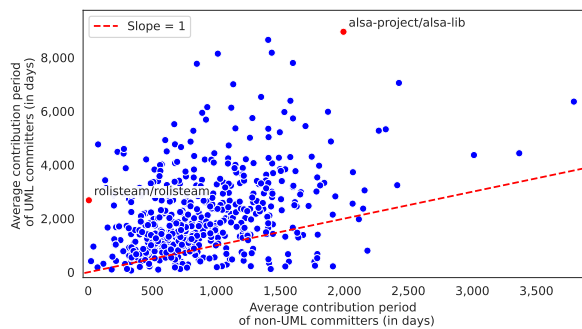


Fig. 6. Average Contribution Periods of UML vs. non-UML Committers

We marked two outlier repositories in red in the scatter plot. The first is the one with the lowest average contribution period of non-UML committers while still having a high average contribution period of UML committers, *rolisteam/rolisteam*.




The contributors for *rolisteam/rolisteam* can be seen in Table IV, with the UML committers marked with the  icon.


TABLE IV
CONTRIBUTORS FOR ROLISTEAM/ROLISTEAM
( UML COMMITTERS)


GitHub Profile Name	Contribution Period	Commits
Renaud Guezennec 	2,682	5,558
Vladar4	56	2
Etienne	8	3
Tyler Schmidt	8	3
Tomaz Canabrava	5	35
Milan Irigoyen	3	4
Paul Brown	0	4
Gissu	0	5
Ben Cooksley	0	4
Yann Escarbassiere	0	3
Patrick José Pereira	0	3
Grégoire Barbier	0	1
IBPX	0	1

We can see the repository has a single maintainer, *Renaud Guezennec*. Most of the remaining contributors had all of their commits on the same day (author contribution period of 0).

The main maintainer is the only person who has also created or modified any UML diagrams, which explains the high average contribution period of UML committers.

The other marked repository has the highest average contribution period of UML committers, *alsa-project/alsa-lib*. Table V shows the top 5 authors by number of commits.

TABLE V
TOP 5 CONTRIBUTORS BY NUMBER OF COMMITS FOR
ALSA-PROJECT/ALSA-LIB ( UML COMMITTERS)

GitHub Profile Name	Contribution Period	Commits
Jaroslav Kysela 	8,962	1,967
Takashi Iwai	8,027	925
Clemens Ladisch	4,764	126
Takashi Sakamoto	2,870	151
Abramo Bagnara	1,227	355

This project is similar to *rolisteam/rolisteam*: Only one contributor modified any UML diagrams. However, in the *alsa-project/alsa-lib* repository, many other contributors have been active for a long time, making numerous non-UML commits.

Number of UML Committers vs. non-UML Committers. We investigate how many contributors in a repository are UML committers. Figure 7 shows a scatter plot of the number of UML versus non-UML committers.

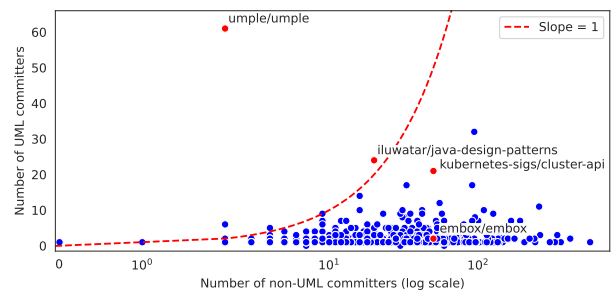


Fig. 7. Number of UML vs. non-UML Committers

In almost every single case, the number of non-UML committers is significantly higher (Mann-Whitney U test $p \leq 0.01$, Cliff's delta $\delta = -0.93$, large effect) than the number of UML committers (repositories under the red dashed line).

We investigated more in detail a few different cases: *iluwatar/java-design-patterns* with a ratio close to 1.0, *umple/umple* with more UML than non-UML committers, *kubernetes-sigs/cluster-api* with more non-UML committers (but still more than 20 UML committers), and *embox/embox*, as representative of the typical case.

The *iluwatar/java-design-patterns* project consists of an educational repository used to teach Java design patterns.

Every pattern follows this template: A Java example, a UML diagram, and a readme with a description and the embedded UML diagram. Any author wanting to add or update a design pattern must do the same for the corresponding UML diagram.

The *umple/umple* project is the repository for the Umple programming language.⁸ As a model-oriented programming language, Umple heavily relies on UML diagrams, justifying it being the perfect outlier.

kubernetes-sigs/cluster-api houses a set of APIs for managing Kubernetes clusters, and keeps a Markdown book which documents the APIs⁹ (e.g., their implementation, usage, extension). The documentation and proposals for new features contain a wealth of UML diagrams to support the text. The diagramming work is uniformly spread among 28 contributors. Each author makes an average of less than 2 UML commits.

The previous three repositories are in stark contrast with the typical repository in our dataset, for which we consider *embox/embox* as an example. This repository has 154 authors, and only 2 have actively worked on UML diagrams. The top 5 contributors by the number of commits in Table VI include both UML committers. This is closer to what we saw in the contribution periods of the *alsa-project/alsa-lib* and *rolisteam/rolisteam* repositories.

TABLE VI
TOP 5 CONTRIBUTORS BY NUMBER OF COMMITS FOR EMBOX/EMBOX
(UML COMMITTERS)

GitHub Profile Name	Commits	UML Commits
Anton Bondarev	5,581	0
Anton Kozlov	3,154	28
Alex Kalmuk	2,811	0
Denis Deryugin	2,584	0
Eldar Abusalimov	2,252	41

Number of Commits by UML vs. non-UML Committers.

We wanted to see whether UML committers contribute more to projects than non-UML committers. We hypothesized that UML committers are more likely to be main contributors: UML committers should be more familiar with the project and more capable of making diagrams; they want developers to be able to contribute easily, so they provide diagrams that help onboard; they want users to use their project and diagrams are a helpful tool to convey how it works.

⁸See <https://cruise.umple.org/umple/>

⁹See <https://cluster-api.sigs.k8s.io/>

In *kubernetes-sigs/cluster-api*, the main contributors were not the main UML diagrammers, as in *embox/embox*. Previously, we analyzed outliers, but UML committers make almost 4 times more commits than non-UML committers (Mann-Whitney U test $p \leq 0.01$, Cliff's delta $\delta = 0.69$, large effect).

In Figure 8, we see a box plot of the average number of commits by UML and non-UML committers.

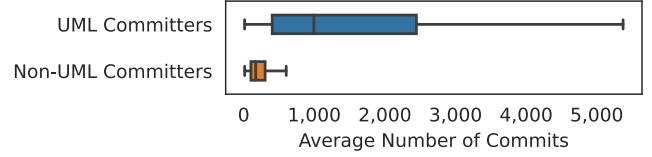


Fig. 8. Average Number of Commits by UML vs. non-UML Committers

Note that we show the box plot without outliers to focus on the general trend, but we found repositories where UML committers make 534 times more commits. These repositories are similar to *rolisteam/rolisteam*, with a single contributor who makes most commits and is the sole UML committer.

Dedicated UML Diagrammers. How often are there dedicated UML diagrammers (i.e., those who modify UML diagrams but do not modify source code)? Looking back at Table IV it seems unlikely that the 5.5k commits of Renaud Guezennec are all UML-related. We analyze UML committers who did not modify source code files.

In Table VII, we show the 13 repositories we found. Only 2.4% of repositories have dedicated diagrammers. Manually analyzing the repositories in the list more in-depth, we see that almost half of them contain little to no source code (🔒).

TABLE VII
DEDICATED DIAGRAMMERS (🔒 LITTLE OR NO SOURCE CODE).

Repository Name	GitHub Profile Name	Commits
<i>adorsys/open-banking-gateway</i>	Dora Nziali	16
<i>adorsys/xs2a</i>	Daria Lavrenova	50
	Olga Levandovska	16
<i>apache/isis</i>	Alexander Schwartz	21
<i>deegree/deegree3</i>	Danilo Bretschneider	26
🔒 <i>edmcouncil/fibo</i>	Mike Bennett	85
	Brent	52
🔒 <i>hyperledger/aries-rfcs</i>	ashcherbakov	41
	Vinomaster	32
🔒 <i>kubernetes/enhancements</i>	Patrick Ohly	61
🔒 <i>openshift/enhancements</i>	Enxebre	17
🔒 <i>progit/progit</i>	Igor Murzov	156
	Anthony Gaudino	24
<i>programmevitam/vitam</i>	Clemence Boyer	64
	edith	32
<i>uportal-project/uportal</i>	Christian Cousquer	21
🔒 <i>w3fj/polkadot-spec</i>	Fabio Lama	1378
<i>wix/detox</i>	wixmobile	343

w3fj/polkadot-spec is a specification repository, *progit/progit* is a book, *edmcouncil/fibo* holds a specification for an ontology, and *kubernetes/enhancements*, *hyperledger/aries-rfcs*, and *openshift/enhancements* are repositories used to discuss enhancements or features for other repositories. *wix/detox* is a source code repository but the author who modified only UML diagrams is a bot account for publishing documentation.

One example of a dedicated diagrammer in a repository with source code is *Daria Lavrenova* in the *adorsys/xs2a* repository. Their GitHub profile shows that they are a program manager (PM). Based on their commit history, which is related to documentation and roadmap planning entirely, it looks like they are doing program management work for *adorsys/xs2a*. *Olga Levandovska* has a similar commit history in the same repository and is indicated as PM in their GitHub profile, so we can assume they also have a similar role.

Dora Nzali of *adorsys/open-banking-gateway*, a different repository of the same owner, has a similar commit history to *Daria* and *Olga*, but no reference to PM activities in the GitHub profile. Still, given the similarity, it looks like *adorsys* has PMs dedicated to updating roadmaps and documentation, including UML diagrams.

In our dataset there are only few software systems where people are dedicated to UML diagramming without contributing to source code, this is indeed a rare occurrence in OSS.

Summarizing: RQ₄ focuses on developers who create and maintain UML artifacts. There are very few UML committers, which supports the idea that UML is not being used to its full potential. If developers in a collaborative development platform contribute almost exclusively to source code and not to UML diagrams, it might imply that they do not perceive UML diagrams as useful as the source code itself. This is just one of the potential causes and it needs further investigation.

RQ₄ – FINDINGS

- F₁ UML committers tend to be among the longer-standing members of their respective projects.
- F₂ UML committers make ~4 times more commits.
- F₃ There are almost never dedicated UML diagrammers. Only 2.4% of repositories have contributors who modify UML diagrams and not source code.

VI. DISCUSSION

The rise and fall of UML cannot be reconstructed by file extensions alone, but through the extensions we could find a reliable source to analyze UML's long-term evolution. UML has never been widely popular in open source software. Our approach led to insights on how the nature of UML support changed from standalone graphical tools, to IDE plugins, to human-readable text. Despite its relative popularity, in this latter form, the graphical quality of the rendered diagram (thus its usability) is still subject to many compromises.

In terms of our considered activity metrics, UML repositories are not significantly different from non-UML ones (except for a slightly higher number of commits). What stands out though is that the majority of them use Java as their main language. We attempted to cluster languages of UML repositories to find commonalities but to no avail. Further research on how UML is used in repositories of different languages might shed light on the features that make UML so appealing for certain languages.

Contributors who modify UML files, on the other hand, are longer-standing and more active members of a project. Although only a small percentage of developers usually contribute to UML, there are exceptions where UML is a collaborative effort (e.g., *kubernetes-sigs/cluster-api*). Nevertheless, although commits are a coarse-grained unit of measure, the number of UML commits per single author is still minimal (less than 4 in *kubernetes-sigs/cluster-api*).

New tools and human-readable formats should consider several aspects to keep this growing trend of UML adoption in OSS development (e.g., layouts, reducing tool fragmentation, separating presentation and content concerns). Focusing on artifacts partially obscures the motivations behind some of the highlighted phenomena. Nevertheless, making these phenomena intelligible and more transparent for the software engineering audience (especially beyond the modeling community) could spark new studies on the *Whys* driving them.

Underutilization. Going back to our initial meta-question, the underutilization of UML is, in our opinion, tied to the lack of standard unified tools. The software market in the early 2000s generated so many alternatives that the fragmentation created a Babel of UML realization dialects, competing with the “official” XMI specification which, in the end, shared XML's fate. Most commercial tools have expensive licensing options, making them unlikely choices for volunteers. A textual representation created and modified in any text editor increases the *pool of potential* contributors. The fact that this *potential* does not result in an *effective* increase indicates that some factors counterbalance this benefit. We argue that one of them is the shortcomings of current tools for human-readable formats (e.g., lack of layouting control).

Reinforcing the economic explanation, there is a focus on *simple* representations without dedicated tooling, where the serialization of the model is the text itself. This agrees with the surveys indicating that UML is more often used with a loose syntax. The goal is not model-driven development, but rather having a “satisficing” (i.e., satisfactory and sufficing) model to support the design phase or to become part of the system's documentation for program comprehension. Nevertheless, in most repositories, only few contributors deal with UML artifacts and those who are not developers are even fewer.

If we do not take into account the shortcomings of the new wave of UML tools and formats, the resurgence wave might not bring UML out of the underutilization condition.

Textual vs. Visual Model. What is given up in the current implementations of text-based UML is the visual power tied to element positioning and layout. We highlighted how core developers usually cover UML creator and maintainer roles. Familiarity with source code may influence their tendency to prefer a textual representation for graphics. Human-readable UML formats do not need to be complicated to be usable for complex systems, but some original features of the visual language must be re-incorporated to maximize effectiveness.

Implicit validation of the model with a non-ambiguous grammar for the language can guarantee syntactically correct models, reducing the burden of correctness for the creator.

Appropriate layout strategies could give back the visual expressiveness and clarity that separates functional from dysfunctional UML diagrams. Finally, we argue that one of the advantages of tools like PlantUML and Mermaid is that they provide a subset of the UML specification in a digestible format. The most useful features are the easiest to use. For example, a class diagram with a few entities is just a few lines of text, moving back the focus on the design and the relationships of composing elements. Automatic layout of elements in a consistent and controlled way will probably be the make or break feature of such tools.

Finally, a reflection on teaching UML in academic courses. Our study informs and justifies the presentation of currently trending tools (*e.g.*, PlantUML, Mermaid) and encourages shifting the emphasis from the graphical to the conceptual model, distilling it into human-readable textual formats.

VII. THREATS TO VALIDITY

Construct Validity. In RQ₁ we investigate UML extensions as a way to capture UML files. The list of UML extensions cannot be exhaustive. Since the resulting list is used throughout our analyses, we mitigate this risk by integrating multiple strategies to obtain the final dataset of repositories and their UML files: We search for popular UML tools and include their supported input and output formats; we search for examples and counter-examples of UML diagrams by analyzing all UML file extensions in our dataset; we apply progressive refinement of our file tagging strategies for each extension until saturation, to be able to capture all UML files.

In our initial dataset, we filter projects based on criteria that are not directly related to the project's level of documentation. While the attitude towards documentation in general can indicate the possibility to find UML, filtering based on such criteria (*e.g.*, a minimum amount of documentation) would misrepresent the actual popularity of UML.

While developers may use temporary diagrams (*e.g.*, sketches for live discussion), they usually do not persist them. We focus on UML artifacts intended for long-term project support, transient ones are out of the scope of our study.

Internal Validity. Discriminating UML and non-UML content in files can be subjective. The first author collected examples and counter-examples of UML files. Dubious cases were discussed among the other authors until a consensus was reached. To further mitigate this risk, we restricted our analysis to the files that could be described non-ambiguously by automatic UML tagging strategies (*e.g.*, regular expressions for file content to identify specific UML "signatures") or exhaustively manually annotated.

External Validity. We extract our dataset from open source projects hosted on GitHub. This is a threat to the generalizability of our results to other types of projects (*e.g.*, closed source developed in a company). When creating our dataset, we perform a trade-off to have a large enough sample while mitigating the risk of underestimating the presence of UML due to simple projects. On the other hand, this filtering strengthens our conclusions on the underutilization of UML.

VIII. CONCLUSION

UML has been taught as the "be-all and end-all" of software design and modeling to generations of students. On the other hand we find practitioners using small subsets of UML at best, often in an informal, imprecise declination. Only a tiny fraction of relevant GitHub projects include any form of UML diagrams. Something in the chain fell apart between academia and practice. The new wave of interest about UML, represented by the recent resurgence of human-readable text-based formats, should be met with prompt reaction. We need to address the shortcomings we foresee, like the diminished value of a diagram with an uncontrollable layout, to leverage the advantages that textual representations bring, for example, ease of parsability, reviewability, and comparability. The timeliness with which the community will increase its awareness of the implications of the recent evolution will hopefully make a difference in the survivability of UML in its new incarnation.

UML is back and popular again but still underutilized. The lack of maintenance that new formats of UML diagrams seem to experience is nothing new for the research community. What is different is that for these types of artifacts we could find better ways to support design and documentation efforts. UML is a precious resource for large software systems and tool support cannot ignore some of the trends we highlighted, first and foremost, the widening gap between repositories merely containing and those actively modifying UML files. In this regard, standalone tools disconnected from the evolving source code can become more of a hindrance than a help.


Finally, projects with and without UML are almost indistinguishable from the surface, but there is a large gap between contributors who work on UML artifacts and those who do not. The lack of dedicated figures responsible of creating, maintaining, and evolving UML diagrams should make us rethink the role of UML closer to developers than to the mythical figure of the software architect. The alternative is to embrace a no-code philosophy augmented by large language models, dreaming again the 2000s dream of round-trip engineering, and using phone pictures of whiteboard sketches as GPT-Xy prompts asking to produce a text-based UML diagram.

ACKNOWLEDGMENTS

This work is supported by the Swiss National Science Foundation (SNSF) through the project "FORCE" (SNF Project Number 232141).

REPLICATION PACKAGE

The replication package, including the list of repositories, the list of importable, exportable, and excluded extensions, the strategies to tag UML files for each extension, the final dataset, and a Jupyter Notebook with the analyses, is available at:

 <https://doi.org/10.6084/m9.figshare.28008434>

REFERENCES

- [1] A. Watson, "Visual modelling: Past, present and future," *White Paper UML Object Modeling Group*, pp. 1–6, 2008. [Online]. Available: https://www.omg.org/UML/Visual_Modeling.pdf
- [2] S. S. Alhir, *UML in a Nutshell: A Desktop Quick Reference*. O'Reilly, 1998.
- [3] G. Booch, *Object-Oriented Analysis and Design with Applications*, 2nd ed. Benjamin-Cummings Publishing Co., Inc., 1993.
- [4] I. Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley, 1992.
- [5] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. E. Lorensen, *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.
- [6] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed. Addison-Wesley, 2018.
- [7] M. Ozkaya and F. Erata, "A survey on the practical use of UML for different software architecture viewpoints," *Information and Software Technology*, vol. 121, p. 106275, 2020.
- [8] T. G. Moreira, M. A. Wehrmeister, C. E. Pereira, J.-F. Petin, and E. Levrat, "Automatic code generation for embedded systems: From UML specifications to VHDL code," in *Proceedings of INDIN 2010 (International Conference on Industrial Informatics)*. IEEE, 2010, pp. 1085–1090.
- [9] P. Hruby, "Specification of workflow management systems with UML," in *Proceedings of OOPSLA 1998 (Workshop on Implementation and Application of Object-oriented Workflow Management Systems)*, vol. 2. ACM, 1998, pp. 1–11.
- [10] A. Kalnins and V. Vitolins. (2006) Use of UML and model transformations for workflow process definitions. arXiv preprint cs/0607044.
- [11] G. De Vito, F. Ferrucci, and C. Gravino, "Design and automation of a COSMIC measurement procedure based on UML models," *Software and Systems Modeling*, vol. 19, no. 1, pp. 171–198, 2020.
- [12] A. M. Fernández-Sáez, D. Caivano, M. Genero, and M. R. V. Chaudron, "On the use of UML documentation in software maintenance: Results from a survey in industry," in *Proceedings of MODELS 2015 (International Conference on Model Driven Engineering Languages and Systems)*. IEEE, 2015, pp. 292–301.
- [13] G. Scanniello, C. Gravino, M. Genero, J. A. Cruz-Lemus, G. Tortora, M. Risi, and G. Doderio, "Do software models based on the UML aid in source-code comprehensibility? Aggregating evidence from 12 controlled experiments," *Empirical Software Engineering*, vol. 23, no. 5, pp. 2695–2733, 2018.
- [14] W. K. G. Assunção, S. R. Vergilio, and R. E. Lopez-Herrejon, *ModelVars2SPL: From UML Class Diagram Variants to Software Product Line Core Assets*. Springer, 2023, ch. in Handbook of Re-Engineering Software Intensive Systems into Software Product Lines, pp. 221–250.
- [15] —, *Reengineering UML Class Diagram Variants into a Product Line Architecture*. Springer, 2023, ch. in UML-Based Software Product Line Engineering with SMarty, pp. 393–414.
- [16] Object Modeling Group. (2024) Unified Modeling Language Specification. OMG. [Online]. Available: <https://www.omg.org/spec/UML>
- [17] M. Ozkaya, "Are the UML modelling tools powerful enough for practitioners? A literature review," *IET Software*, vol. 13, no. 5, pp. 338–354, 2019.
- [18] G. Robles, M. R. V. Chaudron, R. Jolak, and R. Hebig, "A reflection on the impact of model mining from GitHub," *Information and Software Technology*, vol. 164, p. 107317, 2023.
- [19] S. A. Rukmono and M. R. V. Chaudron, "Guiding peer-feedback in learning software design using UML," in *Proceedings of ICSE-SEET 2022 (International Conference on Software Engineering: Software Engineering Education and Training)*. ACM, 2022, pp. 122–133.
- [20] F. Huber and G. Hagel, "Tool-supported teaching of UML diagrams in software engineering education — A systematic literature review," in *Proceedings of MIPRO 2022 (Jubilee International Convention on Information, Communication and Electronic Technology)*. IEEE, 2022, pp. 1404–1409.
- [21] G. Engels, J. H. Hausmann, M. Lohmann, and S. Sauer, "Teaching UML is teaching software engineering is teaching abstraction," in *Proceedings of MODELS-Satellite Events 2005 (Educator's Symposium at the International Conference on Model Driven Engineering Languages and Systems)*. Springer, 2006, pp. 306–319.
- [22] C. F. J. Lange, M. R. V. Chaudron, and J. Muskens, "In practice: UML software architecture and design description," *IEEE Software*, vol. 23, no. 2, pp. 40–46, 2006.
- [23] W. J. Dzidek, E. Arisholm, and L. C. Briand, "A realistic empirical evaluation of the costs and benefits of UML in software maintenance," *IEEE Transactions on Software Engineering*, vol. 34, no. 3, pp. 407–432, 2008.
- [24] T. C. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: The state of the practice," *IEEE Software*, vol. 20, no. 6, pp. 35–39, 2003.
- [25] E. Tryggeseth, "Report from an experiment: Impact of documentation on maintenance," *Empirical Software Engineering*, vol. 2, no. 2, pp. 201–207, 1997.
- [26] R. Jolak, M. Savary-Leblanc, M. Dalibor, J. Vincur, R. Hebig, X. L. Pallec, M. Chaudron, S. Gérard, I. Polasek, and A. Wortmann, "The influence of software design representation on the design communication of teams with diverse personalities," in *Proceedings of MODELS 2022 (International Conference on Model Driven Engineering Languages and Systems)*. ACM, 2022, pp. 255–265.
- [27] E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche, "The impact of UML documentation on software maintenance: An experimental evaluation," *IEEE Transactions on Software Engineering*, vol. 32, no. 6, pp. 365–381, 2006.
- [28] G. Scanniello, C. Gravino, M. Genero, J. A. Cruz-Lemus, and G. Tortora, "On the impact of UML analysis models on source-code comprehensibility and modifiability," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 2, pp. 1–26, 2014.
- [29] C. Gravino, G. Scanniello, and G. Tortora, "Source-code comprehension tasks supported by UML design models: Results from a controlled experiment and a differentiated replication," *Journal of Visual Languages and Computing*, vol. 28, pp. 23–38, 2015.
- [30] Q. Chen, J. Grundy, and J. Hosking, "An e-whiteboard application to support early design-stage sketching of UML diagrams," in *Proceedings of HCC 2003 (Symposium on Human Centric Computing Languages and Environments)*. IEEE, 2003, pp. 219–226.
- [31] R. Dachsett, M. Frisch, and E. Decker, "Enhancing UML sketch tools with digital pens and paper," in *Proceedings of SoftVis 2008 (Symposium on Software Visualization)*. ACM, 2008, pp. 203–204.
- [32] S. Baltes and S. Diehl, "Sketches and diagrams in practice," in *Proceedings of FSE 2014 (International Symposium on Foundations of Software Engineering)*. ACM, 2014, pp. 530–541.
- [33] G. Bergström, F. Hujainah, T. Ho-Quang, R. Jolak, S. A. Rukmono, A. Nurwidyantoro, and M. R. V. Chaudron, "Evaluating the layout quality of UML class diagrams using machine learning," *Journal of Systems and Software*, vol. 192, p. 111413, 2022.
- [34] O. Badreddin, T. C. Lethbridge, and M. Elassar, "Modeling practices in open source software," in *Proceedings of OSS 2013 (Open Source Software: Quality Verification)*. Springer, 2013, pp. 127–139.
- [35] R. Hebig, T. Ho-Quang, M. R. V. Chaudron, G. Robles, and M. A. Fernandez, "The quest for open source projects that use UML: Mining GitHub," in *Proceedings of MODELS 2016 (International Conference on Model Driven Engineering Languages and Systems)*. ACM, 2016, pp. 173–183.
- [36] G. Robles, T. Ho-Quang, R. Hebig, M. R. V. Chaudron, and M. A. Fernandez, "An extensive dataset of UML models in GitHub," in *Proceedings of MSR 2017 (International Conference on Mining Software Repositories)*. IEEE, 2017, pp. 519–522.
- [37] 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 APSEC 2014 (Asia-Pacific Software Engineering Conference)*, vol. 1. IEEE, 2014, pp. 399–406.
- [38] A. Forward, O. Badreddin, and T. C. Lethbridge, "Perceptions of software modeling: A survey of software practitioners," in *Proceedings of C2M: EEMDD 2010 (Workshop from Code Centric to Model Centric: Evaluating the Effectiveness of Model Driven Development)*, 2010.
- [39] T. Ho-Quang, R. Hebig, G. Robles, M. R. V. Chaudron, and M. A. Fernandez, "Practices and perceptions of UML use in open source projects," in *Proceedings of ICSE-SEIP 2017 (International Conference on Software Engineering: Software Engineering in Practice Track)*. IEEE, 2017, pp. 203–212.
- [40] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in GitHub for MSR studies," in *Proceedings of MSR 2021 (International Conference on Mining Software Repositories)*. IEEE/ACM, 2021, pp. 560–564.
- [41] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The promises and perils of mining GitHub," in *Proceedings of MSR 2014 (Working Conference on Mining Software Repositories)*. ACM, 2014, pp. 92–101.

- [42] S. Zhou, B. Vasilescu, and C. Kästner, “How has forking changed in the last 20 years? A study of hard forks on GitHub,” in *Proceedings of ICSE 2020 (International Conference on Software Engineering)*. ACM, 2020, pp. 445–456.
- [43] A. Danial. (2021) cloc: v1.92. Zenodo. [Online]. Available: <https://doi.org/10.5281/zenodo.5760077>
- [44] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, “Mining email social networks,” in *Proceedings of MSR 2006 (International Workshop on Mining Software Repositories)*. ACM, 2006, pp. 137–143.
- [45] E. Kouters, B. Vasilescu, A. Serebrenik, and M. G. J. Van Den Brand, “Who’s who in Gnome: Using LSA to merge software repository identities,” in *Proceedings of ICSM 2012 (International Conference on Software Maintenance)*. IEEE, 2012, pp. 592–595.
- [46] M. Goeminne and T. Mens, “A comparison of identity merge algorithms for software repositories,” *Science of Computer Programming*, vol. 78, no. 8, pp. 971–986, 2013.
- [47] C. Gote and C. Zingg, “gambit — An open source name disambiguation tool for version control systems,” in *Proceedings of MSR 2021 (International Conference on Mining Software Repositories)*. IEEE, 2021, pp. 80–84.