UML Usage in Open Source Software Development : A Field Study

Hafeez Osman[1] and Michel R.V. Chaudron[1,2]

[1] Leiden Institute of Advanced Computer Science, Leiden University, the Netherlands
hosman@liacs.nl
[2] Joint Department of Computer Science and Engineering Chalmers University of Technology and Goteborg University, Sweden
chaudron@chalmers.se

Abstract. UML is the de facto standard for modeling software designs and is commonly used in commercial software development. However, little is known about the use of UML in Open-source Software Development. This paper evaluates the usage of UML modeling in ten open-source projects selected from common open-source repositories. We evaluated the usage of UML diagrams based on the information available in the open-source projects repositories. Our study covers the types of UML diagrams that are used and how frequently UML models are updated. Our findings also include the change in focus on types of diagram used over time and researches on how the size of models relates to the size of the implementation.

Keywords: UML, Open-source Software Development, Reverse Engineering

1 Introduction

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 [11]. For commercial software development, the use of UML is commonly prescribed as part of a company-wide software development process while in open-source software development (OSSD), there is typically no mandate on the use of UML. Only if the community of developers of the OSSD feels needs (e.g. for their communication) then UML diagrams are produced. Even though some open-source projects employ UML diagrams, these diagrams do not completely correspond with the implementation code. For instance, the number of classes used in class diagrams is typically less than the number of classes that exist in the implementation source code. The usage of UML class diagrams also varies across projects. Almost all OSSD projects that use UML choose to produce class diagrams. Some projects also constructed other types of UML diagrams such as use case diagrams, sequence diagrams and activity diagrams.

One of the benefits of UML is to ease communication between software developers. The nature of OSSD is that software developers normally communicate

with each other using some online communication medium (e.g discussion forum, e-mail, IRC) rather than through physical interaction. There is an anecdotal belief that UML is rarely used in OSSD. However, there is no quantitative research to prove this perception. In this paper, we aim at evaluating the usage of UML diagrams in OSSD projects. We want to discover how UML is used in OSSD without the influence of the stakeholders or users of the system. We explore the publicly software documentation to answer the following questions: 1) What types of UML diagrams are used? 2) How does the size of the design relates to the size of the implementation? and 3) How does timing of changes in the implementation relate to changes in UML diagrams/documentation?

The paper is structured as follows: Section 2 discusses related work. Section 3 describes the case studies used in this research. Section 4 describes the study approach while Section 5 presents the results and findings. This is followed by our conclusion and future work in Section 6.

2 Related Work

Dobing and Parsons [7] performed a survey to find out to what extent the UML is used and for what purpose, differences in the level of diagram use and how successful UML usage is for communication in a team. The research found that the most used types of UML diagrams were use case diagram and class diagram while collaboration diagram was used the least. Dobing and Parsons also conducted another survey to discover the current practice in the use of UML in [8]. The findings of this survey highlighted that the most used UML diagrams were class diagram, use case diagram and sequence diagram.

Grossman et. al [9] performed a study to research the perspective of individuals using UML using the task-technology fit model. This study also addressed the characteristics that affect the usage of UML. Similar to [7] and [8], the result of the most three important diagrams in ranking are use case, class and sequence Diagram. Those studies also found out that it is difficult to discover whether UML provides too much detail or too little detail because it depends on the software technology (i.e. Enterprise System, Web-based system, real-time system) that requires UML to be tailored to the environment.

Yatani et. al [10] conducted an evaluation on the use of diagramming for communication among OSSD developers and also performed a semi-structured interview with developers from a large OSSD project. This study highlighted diverse type of diagram which is used for the communication between the contributors of the system. Not all diagrams used for communication purposes were updated during the project. The study extended by Chung et. al [12] with a survey participated by 230 OSSD developers and designers. For the frequency of updating designs, even though 76% agree that diagrams have value, only 27% practice diagramming very often or all the time during the software development.

Most of the related works use surveys to discover the usage of UML diagram. These surveys are based on the UML practitioners' perspective of how they use UML. In contrast, our study evaluates the use of UML modeling in

OSSD projects by mining the project documentation. Hence, this reflects the real artifacts produced by using the UML notation.

Table 1. List of Case Studies

Project	Description	No. of Releases	URL Source
Maze	Maze-solver is a Micro-Mouse maze editor and simulator.	2	http://code.google .com/p/maze-solver/
JavaClient	The project allows development of applications for Player/Stage using the Java programming language.	3	http://java-player. sourceforge.net/
xUML-Compiler (xUML)	xUml-Compiler takes a user specified data model and associated state machines and produces an executable and testable system.	13	http://code.google .com/p/xuml-compiler/
JPMC	Java Portfolio Management Component (JPMC) is a collection of portfolio management component.	1	http://jpmc.source forge .net/
Neuroph	Lightweight Java neural network framework to develop common neural network architectures.	9	http://neuroph .sourceforge .net/
Gwt-portlets	Free open-source web framework for building GWT (Google Web Toolkit) applications.	6	http://code.google .com/p/gwtportlets/
Wro4J	The project purpose is to improve web application page loading time.	3	http://code.google .com/p/wro4j/
JGAP	Genetic Algorithms and Genetic Programming package.	8	http://jgap.sourceforge .net/
ArgoUML	An open-source UML modeling tool and include support for all standard UML 1.4 diagrams.	19	http://argouml. sourceforge.net
Mars Simulation	Free software project to create a simulation of future human settlement of Mars.	26	http://mars-sim. sourceforge.net/

3 Case Studies

Based on researches by Hutchinson et. al [1], Dobing and Parsons [7], and Erickson et. al [2], we know that the most used UML diagram is the class diagram. Therefore, we performed a search for UML class diagram images using the Google search engine. In particular, we targeted our search on four open-source repositories: SourceForge, Google Code, GitHub and BerliOS. The main keyword used for the search was “class diagram”. Based on the hits of these searches, we browsed the project repositories. Our initial list of candidate cases consisted of 57 projects. We refined the selection by using the following criteria:

- The project should have UML diagrams and corresponding source code (project that have multiple versions is preferred)
- The source code should be written in Java

Projects that are developed in Java is selected because we need to reverse engineer the source code to class diagram for analysis purposes. The reverse engineering tools that we used for this study performs best with Java source codes. We found ten software projects which are suitable that are listed in Table 1.

4 Approach

This section describes the approach we used in this study. We had three main activities in order to answer the following research questions :
RQ1 : What types of UML diagrams are used? Based on the project repository, we manually browsed the documentation and other provided information to find all the UML diagrams that were used in the project.
RQ2 : How does the size of the design relate to the size of the implementation? Our aim was to use one single tool for counting classes of both the design and implementation. Furthermore, for source code, we only wanted to count classes that were actually *designed* for the project's system, hence to exclude library classes that are imported, and would typically not be modeled. To this end, source codes were reverse engineered using several Computer Aided Software Engineering (CASE) tools i.e MagicDraw 17.0[3] and Enterprise Architect 7.5[4]. The reverse engineered design was then exported to XML Metadata Interchange (XMI) files. From the resulting XMI files, software design metrics were computed using the SDMetrics [5] tool and all library classes were removed.
RQ3 : How does timing of changes in the implementation relates to changes in UML diagrams/documentation? For source code, we manually extracted the dates of releases from the project repositories. For UML diagrams, we looked at the date-information provided by the system documentation, developer's manual and other related documents in the project repository.

5 Results and Findings

In this section, we present the results. We group our results into the three questions presented in the previous section.

5.1 Usage of UML Diagrams

The UML diagram that was mostly used in our set of open-source projects is the class diagram. This is to be expected because our main keyword of searching

[3] http://www.nomagic.com/
[4] http://www.sparxsystems.com.au/
[5] http://www.SDMetrics.com/

for case study was based on class diagrams. Table 2 shows which other types of diagrams were used. The term 'yes' in Table 2 means that the project used at least one instance of a UML diagram specified in the table. The use of UML in OSSD projects seems driven by a need to codify high level knowledge. For example, ArgoUML did not use sequence diagrams in their modeling until there was a new feature. Only this new feature was explained by a sequence diagram. In general, the case studies show that the most used UML diagrams in OSSD are use case, component, package, class, sequence/interaction and activity diagram. The following subsections describe the results in more detail.

Table 2. UML Diagram Usage

			Structure Diagram					Behaviour Diagram		
No	Project	Use Case	Component Diagram	Package Diagram	Class Diagram	Composite Structure Diagram	Object Diagram	Sequence/ Interaction Diagram	Activity Diagram	State Machine Diagram
1	Maze	No	No	No	Yes	No	No	No	No	No
2	JavaClient	No	No	No	Yes	No	No	No	No	No
3	xUML	No	No	No	Yes	No	No	No	No	No
4	JPMC	Yes	No	Yes	Yes	No	No	No	No	No
5	Neuroph	No	No	No	Yes	No	No	No	No	No
6	Gwt-portlets	No	No	No	Yes	No	No	Yes	No	No
7	Wro4J	No	No	No	Yes	No	No	No	No	No
8	JGAP	No	No	No	Yes	No	No	No	No	No
9	ArgoUML	No	Yes	Yes	Yes	No	No	Yes	Yes	No
10	Mars	No	Yes	No	Yes	No	No	No	Yes	No

Use Case Diagram. Use case diagrams were used by only one OSSD project: JPMC (see Table 2). A use case diagram is used to describe the desired functionality of the software product [3]. Most of these OSSD projects have specified their goal but the specification and the interaction between the user and system were explained in text.

Component Diagram. Component diagrams are used to divide the system into components and show their interrelationships through the breakdown of components into a lower-level structure [5]. ArgoUML provided one component diagram from an old design document to illustrate the interaction between early developed component and packages. The Mars Simulation project provided two component diagrams i.e. 1)'Top Level Diagram' illustrated dependencies between three components, and 2)'Simulation Component Diagram' illustrated more details about the relationship between a simulation component and other related components.

Package Diagram. Package diagrams provide a grouping construct that allows to group design elements together into higher-level units [5]. The JPMC project presents almost all main packages and their dependencies in a package diagram. Meanwhile, ArgoUML presented two package diagrams. The first package diagram in this project illustrated the dependencies between packages with

two other package representing external libraries. The second package diagram illustrated the high level package in this project.

Class Diagram. Class diagram is the most used diagram type in these case studies. Most of the case studies only show classes that are important in the system. The correspondence between classes and implementation is discussed in section 5.2.

Sequence/Interaction Diagram. Sequence diagrams were used by two OSSD projects. However, both projects have only one sequence diagram per project. ArgoUML introduced a sequence diagram after eight version releases. Table 4 shows that only after version 0.26, a sequence diagram was introduced in the documentation. Perhaps, it is difficult to construct the sequence diagram for the entire release hence, the developer of this project used sequence diagram for a new feature. The gwt-portlets project used only one sequence diagram. We assume that the described flow contains crucial information for the system because the classes that were involved in the sequence diagram were presented in a class diagram that shows the key classes of the system.

Activity Diagram. Activity diagramming or activity modeling emphasizes the flow and conditions for coordinating lower-level behaviours [4]. This study found that there were two OSSD projects used the activity diagram. However, not all activity diagrams in these projects are related to the software development. ArgoUML used an activity diagram to present the flow of managing issues in the project. Meanwhile, the Mars Simulation project used one activity diagram for specifying a specific feature of the project.

Table 3. Classes in Design versus Classes in Implementation

No	Project	No. of Classes in Design	No. of Classes in Implementation	% of Design in Implementation
1.	Maze	28	69	40.58
2.	JavaClient	57	215	26.51
3.	xUML	45	172	26.16
4.	JPMC	24	126	19.05
5.	Neuroph	24	179	13.41
6.	gwt-Portlets	20	178	11.24
7.	Wro4J	11	100	11.00
8.	JGAP	18	191	9.42
9.	ArgoUML	33	909	3.63
10.	Mars Simulation	31	953	3.25
	Total	291	3092	16.43

5.2 Ratio between Design and Implementation

This subsection presents the results of analyzing the ratio between classes in the design and classes in the implementation. Since there are multiple versions of both the design and implementation in most of the case studies, we chose a pair

with a high ratio of design to implementation. For example, for the Neuroph project, we selected version 2.3 because this version has a high number of designed classes due to the fact that the project starts updating UML diagrams at this point in time. The Maze project has the highest ratio of classes in design to classes in implementation. This is a relatively small project which consisted of 69 classes in the implementation and 40 % of these classes were represented in the UML design. In absolute numbers, the highest number of classes in a design is found in the JavaClient project with 57 classes.

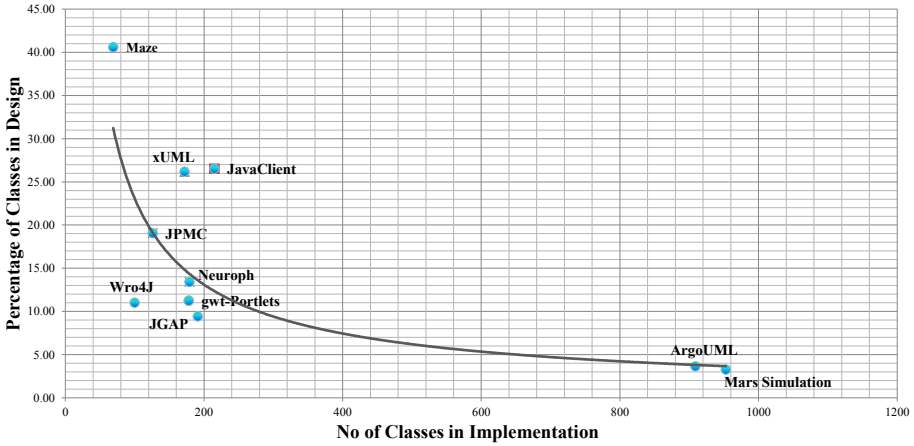

Fig. 1. Classes in Design vs Classes in Implementation

5.3 Frequency of Updating UML models

This subsection presents the frequency of updating the UML models of the case studies. We would like to know whether UML diagram is used throughout the projects or only in initial phases. We analysed the case studies that have multiple versions of releases to assess the frequency of updating the diagrams while the systems evolve through subsequent releases. Even though there were multiple versions of system releases for the Mars Simulation, JavaClient, JPMC, Gwt-Portlet, Maze and xUML-compiler project, their UML diagrams were not changed. For instance, the Mars Simulation project has released 26 versions of source code. The UML designs were only uploaded in Dec 2009. Based on that date, we assume that this design corresponds with this release version 2.87 and above. This indicates that the earlier 19 versions of the software did not have a UML model. However, we could not disregard the fact that the design may be created earlier than the date it was uploaded.

The result also shows that the frequency of updating UML diagram is low. In most of the case studies, a new UML diagram is created when there is a new feature of the system introduced in a new version or release. Only the Neuroph and ArgoUML project actually modified existing diagrams. Other project only added

new diagrams to their documentation, but did not modify previously existing diagrams. In the ArgoUML project, we found that there was an increasing amount of diagrams at the same time as the number of project contributors increases. The work by Wen Zhang et. al [6] shows that there was an increasing amount of participants in version 0.26. The ArgoUML project updated and added a lot of UML diagrams in version 0.26. We hypothesize that the documentation was elaborated to cater for a group of newcoming developers that are looking for information about the design.

Next, we discuss the ArgoUML project as an example of a project that did update their UML designs across multiple versions of releases. Table 4 shows which types of diagrams were used across subsequent versions over time. The table shows in early stages of development, diagrams were made that represent the high level structure of the system (component, package and class). As development time progresses, diagrams are added that represent the dynamic behaviour of the system through activity diagrams (v 0.16) and sequence diagrams (v 0.26). Also, at the later stages of development, component diagrams are no longer used. We believe this trend to be typical of the use of modeling in software development in general : Firstly, the developers design the overall structure and later continue to flesh out behavioural aspects of the design. Figure 2 shows the evolution of UML Diagrams in every versions of release. Figure 2 also shows the evolution of the number of classes. It is explicitly shown that the UML diagrams are created in the early stage of software release and then updated occasionally.

Table 4. List of UML Diagrams used in ArgoUML Project

No	Release Version	Date	Source	Component Diagram	Package Diagram	Class Diagram	Activity Diagram	Sequence/ Interaction Diagram
1	0.10.1	09.10.2002	Old Design Document	Yes	Yes	Yes	No	No
2	0.12	18.08.2003	Cookbook 2003 and Old Design Document	Yes	Yes	Yes	No	No
3	0.14	05.12.2003	Cookbook 2003 and Old Design Document	Yes	Yes	Yes	No	No
4	0.16	19.07.2004	Cookbook-0.16	No	Yes	Yes	Yes	No
5	0.18.1	30.04.2005	Cookbook-0.18.1	No	Yes	Yes	Yes	No
6	0.20	09.02.2006	Cookbook-0.20	No	Yes	Yes	Yes	No
7	0.22	08.08.2006	Cookbook-0.22	No	Yes	Yes	Yes	No
8	0.24	12.02.2007	Cookbook-0.24	No	Yes	Yes	Yes	No
9	0.26	27.09.2008	Cookbook-0.26	No	Yes	Yes	Yes	Yes
10	0.26.2	19.11.2008	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
11	0.28	23.03.2009	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
12	0.28.1	16.08.2009	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
13	0.30	06.03.2010	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
14	0.30.1	06.05.2010	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
15	0.30.2	08.07.2010	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
16	0.32	28.01.2011	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
17	0.32.1	23.02.2011	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
18	0.32.2	03.04.2011	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes
19	0.34	15.12.2011	Cookbook-0.26.2	No	Yes	Yes	Yes	Yes

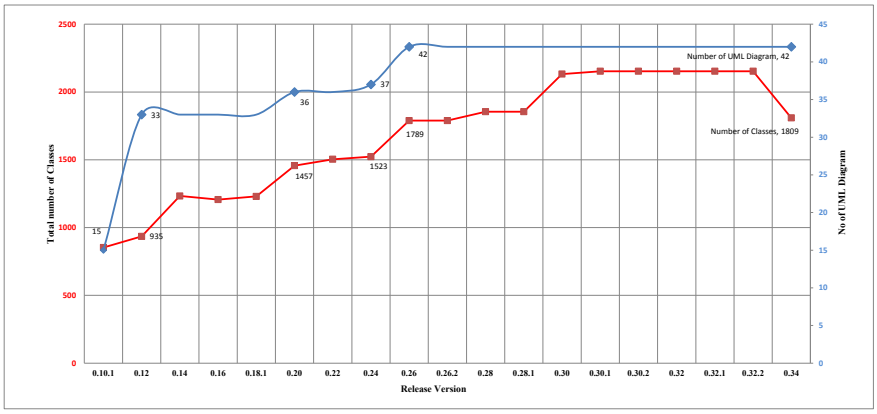

Fig. 2. ArgoUML Evolution in UML Diagrams and Number of Classes

5.4 Threats to Validity

This section describes the threats to validity of this study. In term of case study selection, there could be more case studies if we included more open-source repository. The selected projects may not represent all the OSSD because the selected case studies can be considered as small and medium type of system development. In addition, we also did not have projects with the number of classes between 250 and 800. The result could be different if more larger projects would be included in this study. The study was done based only on using the information in the project repositories and also the projects websites. It may be the case that developers use UML in their communication or for internal use without uploading their diagrams in the project repository. This study also only used the date listed as the upload-date of the documents in the repositories. The document may be created far before the uploaded date. Thus, the matching of the date of documentation and the version may not be accurate.

6 Conclusion and Future Work

This study aims to discover the usage of UML diagram in the OSSD projects. To this end, ten case studies were collected from online repositories. Three main questions were studied: What types of UML diagrams are used?, How does the size of the design relates to the size of the implementation? and How does timing of changes in the implementation relate to changes in UML diagrams/documentation? By studying the evolution of UML models across versions, we found that the focus of modeling shifts from structural aspects in the early phases of development, to dynamic behaviour in the later stages of development. The frequency of updating UML models is low. We found two triggers for updating UML diagrams: 1) if there are changes in the features of the system,

and 2) if there is a group of newcomers joining the project. The latter cause confirms the role of UML models as a way of codifying design knowledge for communicating the design. Overall, this paper shows that open-source projects can be used as empirical source for studying usage of UML modeling.

For future work, it would be interesting to extend this study by performing a broader survey or interview OSSD developers to find out the reasons for or against using UML diagrams in their development. Also, it is interesting to ask developers after their pattern in updating UML models. Finding more case studies and extending the case studies to languages other than Java. This would allow to differentiate results between programming languages.

References

1. J. Hutchinson, J. Whittle, M. Rouncefield and S. Kristoffersen, "Empirical Assessment of MDE in Industry," Proc. of the 33rd International Conference on Software Engineering(ICSE '11), pp. 471–480. ACM New York (2011)
2. Erickson, J., Siau, K.:Theoretical and Practical Complexity of Modeling Methods, In: Communications of the ACM, Vol. 50 Issue 8, pp. 46–51. ACM New York (2007)
3. Grechanik, M., McKinley, K.S., Perry, D.E.: Recovering And Using Use-Case-Diagram-To-Source-Code Traceability Links, In: ESEC-FSE '07 Proc. of the 6th joint meeting of the European softw. eng. conference and the ACM SIGSOFT symposium on The foundations of softw. eng., pp. 95–104 ACM New York (2007)
4. Object Management Group(OMG): Unified Modeling Language Specification, Superstructure Version 2.4.1, `http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/`,pp. 303, (2011)
5. Fowler, M.: UML Distilled 3rd Edition. A Brief Guide to the Standard Object Modeling Language, pp.89, 141. Addison-Wesley (2004)
6. Zhang, W., Yang, Y. Wang, Q.: Network Analysis of OSS Evolution: an Emperical Study on ArgoUML Project. In: IWPSE-EVOL '11 Proc. of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution, pp. 71–80, ACM New York (2011)
7. Dobing, B., Parsons,J.: How UML is used, In: Communications of the ACM, Vol. 49, Issue 5, pp. 109–113, ACM New York (2006)
8. Dobing, B., Parsons,J.: Current Practice in the Use of UML, In: Proceeding of ER 2005 Workshops AOIS, BP-UML, CoMoGIS, eCOMO, and QoIS, pp.2–11, Springer-Verlag Berlin Heidelberg (2005)
9. Grossman, M., Aronson, J.E., McCarthy, R.V.: Does UML make the grade? Insights from the software development community, In: Information and Software Technology, vol. 47, Issue 6, pp 383–397, Elsevier (2005)
10. Yatani, K. Chung, E., Jensen, C., Truong, K.N.: Understanding how and why open source contributors use diagrams in the development of Ubuntu, In: CHI '09 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 995–1004, ACM New York (2009)
11. Grady, B., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide, pp.72, Addison Wesley (1998)
12. Chung, E., Jensen, C., Yatani, K., Kuechler, V., Truong, K.N.: Sketching and Drawing in the Design of Open Source Software, In: VLHCC '10 Proc. of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 195–202, IEEE Computer Society, USA (2010)