Interactive scalable condensation of reverse engineered UML class diagrams for software comprehension

Osman, M.H.B.

Chapter 3

UML Usage in Open Source Software Development

UML is the 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 chapter evaluates the usage of UML modeling in ten open-source projects selected from common open-source repositories. It covers the types of UML diagrams that are used, the level of detail that is applied, and the frequency of updating UML models. Our findings also include the application of UML modeling at different levels of detail for different purposes, the change in focus on types of diagram used over time, and research on how the size of models relates to the size of the implementation.

3.1 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 [32]. 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 the need (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 to the

This chapter is a more detailed version of a publication entitled **"UML Usage in Open Source Software Development : A Field Study"**, In Proceedings of the 3rd International Workshop on Experience and Empirical Studies in Software Modelling (EESSMod 2013)

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 face-to-face 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 chapter, we aim at evaluating the usage of UML diagrams in OSSD projects. We want to investigate how UML is used in OSSD without the influence of the stakeholders or users of the system. We assume that the UML diagrams that exist in a project document means such diagrams are used in the project. The reason is that when the cost (effort) is spent in developing an artifact, such artifact should be used and provides a benefit in the development or maintenance phase [187].

We explore the publicly available software documentation to answer the following questions: 1) What types of UML diagrams are used? 2) How does the size of the design relate to the size of the implementation? 3) What level of detail is used in UML diagrams? and 4) How does timing of changes in the implementation relate to the changes in UML diagrams/documentation?

The chapter is structured as follows: Section 3.2 discusses related work. Section 3.3 describes the case studies used in this research. Section 3.4 explains the study approach while Section 3.5 presents the results and findings. This is followed by our conclusion and future work in Section 3.6.

3.2 Related Work

Dobing and Parsons [48] performed a survey to find out to what extent UML is used and for what purpose, what are the differences of the levels of detail used and how successful UML usage is for communication in a team. The survey was conducted using a web survey and participated by 171 UML practitioners. The research found that the most used types of UML diagrams were use case diagrams and class diagrams while collaboration diagrams were used the least. In [47], Dobing and Parsons also conducted another survey to investigate the current practice in the use of UML. There were 299 responses in the survey (with the endorsement of the Object Management Group (OMG) [119]). The findings of this survey highlighted that the most used UML diagrams were class diagrams, use case diagrams and sequence diagrams. This research also discovered that class diagrams and sequence diagrams play a major role in specifying system requirements for programmer, documenting the design for future

maintenance and in clarifying understanding of the application among team members.

Grossman et al. [67] performed a study on the individual perspective of using UML. This study also addressed the characteristics that affect the usage of UML. Similar to [47] and [48], the result of the most important diagrams in ranking are use case diagrams, class diagrams and sequence diagrams. Those studies also found out that it is difficult to determine 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). The study suggested that UML diagrams need to be customized based on the environment.

Yatani et al. [182] conducted an evaluation on the use of diagramming for communication among OSSD developers and also performed semi-structured interview with developers from a large OSSD project. This study highlighted a diverse types of diagrams that is used for the communication between the contributors of the system. Not all diagrams used for communication purposes were updated during the project.

Chung et al. [42] carried out a survey that was participated by 230 OSSD developers and designers. Their findings demonstrate that 1) In terms of frequency of updating designs, even though 76% agree that diagrams have value, only 27% practice diagramming very often or all the time, and 2) The UML diagrams are only used for formal documentation purposes.

Most of the related works use surveys to explore the usage of UML diagrams. These surveys are based on the 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.

3.3 Case Study

One of the challenges of this study was to find suitable OSSD Projects that use UML diagrams. Based on research by Hutchinson et al. [81], Dobing and Parsons [48], and Erickson et al. [54], we know that one of the most used UML diagrams is the class diagram. For this reason, we performed a search for UML class diagram images using the Google[3] search engine. In particular, we targeted our search on four open source repositories: SourceForge[7], GoogleCode[4], GitHub[8] and BerliOS[2]. The primary keyword used for the search was "Class Diagram". Based on the hits of these searches, we browsed the project repositories to assess their suitability for inclusion in this study. Our initial list of candidate cases consisted of 57 projects (see Appendix A.1). We refined the selection of the case study by using the following criteria:

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

- The amount of classes (in the source code) > 50 classes.

The reason for selecting projects in Java was that we intended to reverse engineer the source code to class diagrams for analysis purposes. The reverse engineering tool that we used for this study performs best with Java source code. We refine our selection of case studies as follows:

1. *Round 1*: Out of 57 projects, we eliminate 21 projects that developed using C++, C#, Pascal, Phyton, etc. (other than Java). Only 36 projects remain to be the candidate.
2. *Round 2*: Discard 13 projects due to the number of classes below 50.
3. *Round 3*: Discard 6 projects because we prefer projects that have more than one version.
4. *Final Round*: In total, 18 projects qualify for the final round. In this round, we thoroughly explore the case studies artifacts (source code, class diagram, documentation). As a result, we found that ten projects that are suitable for our research. Most of the projects we discarded because the projects only provide the latest source code in the repository, even though the projects have several versions of releases (also class diagram).

The list of case studies is shown in Table 3.1. The total numbers of classes involved in these case studies range from 50 to 2000.

3.4 Approach

This section describes the approach we used in this study. We conducted four 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 about the software 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 the implementation. Furthermore, for source code, we only wanted to count classes that were actually *designed* for the project's system, hence we exclude library classes (also test-classes) that are imported, and would typically not be modeled. To this end, source codes are reverse engineered (into class diagrams) using several CASE tools. The CASE tools used in this study were MagicDraw [9] version 17.0 and Enterprise Architect [153] version 7.5. The reverse engineered design was then exported to XML Metadata

Table 3.1: *List of Case Studies*

Project	Description	No. of Releases	URL Source
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/
JavaClient	The project allows development of applications for Player/Stage using the Java programming language.	3	http://java-player.sourceforge.net/
JGAP	Genetic Algorithms and Genetic Programming package.	8	http://jgap.sourceforge.net/
Neuroph	Lightweight Java neural network framework to develop common neural network architectures.	9	http://neuroph.sourceforge.net/
JPMC	Java Portfolio Management Component (JPMC) is a collection of portfolio management components.	1	http://jpmc.sourceforge.net/
Wro4J	It stands for Web Resource Optimizer for Java. The project purpose is to improve web application page loading time.	3	http://code.google.com/p/wro4j/
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/
Maze	Maze-solver is a Micro-Mouse maze editor and simulator.	2	http://code.google.com/p/maze-solver/
Gwt-portlets	Free open source web framework for building GWT (Google Web Toolkit) applications.	6	http://code.google.com/p/gwt-portlets/

Interchange (XMI) files. These were loaded into a UML case tool in which we manually removed all library classes. From the resulting XMI files, software design metrics were computed using the SDMetrics [180] tool.

Table 3.2: *Levels of Detail in UML models*

No	Class Diagram Elements	Low LoD	High LoD
1	Classes (box and name)	YES	YES
2	Attributes	NO	YES
3	Types in Attributes	NO	YES
4	Operations	NO	YES
5	Parameters in Operations	NO	YES
6	Associations	YES	YES
7	Association Directionalities	NO	YES
8	Association Multiplicities	NO	YES
9	Aggregations	YES	YES
10	Compositions	YES	YES

RQ3: What level of detail is used in UML diagrams?

The level of detail (LoD) for all UML diagrams gathered from the projects' repositories was analysed using the level of detail that was defined by Fernández-Sáez et al. [60] (as illustrated in Table 3.2). In addition, we also analyzed the diagrams to identify the technique of constructing the UML diagram (forward or reverse engineering). The UML diagrams were identified as RE-CD if they satisfy the symptoms (or weaknesses) mentioned by Osman [125]. These tasks were done manually.

RQ4: How does timing of changes in implementation relate to the 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.

3.5 Results and Findings

This section describes the result of this study. The results are grouped by the research questions mentioned in the previous section.

3.5.1 Usage of UML Diagrams

The UML diagram that was mostly used in our set of OSSD projects is the class diagram. This was to be expected because our main keyword of searching for the case study was based on class diagrams. Table 3.3 shows which other types of diagrams were

Table 3.3: *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 (6)	No	No	No	No	No
2	JavaClient	No	No	No	Yes (1)	No	No	No	No	No
3	xUML	No	No	No	Yes (1)	No	No	No	No	No
4	JPMC	Yes (1)	No	Yes (1)	Yes (4)	No	No	No	No	No
5	Neuroph	No	No	No	Yes (3)	No	No	No	No	No
6	Gwt-portlets	No	No	No	Yes (3)	No	No	Yes (1)	No	No
7	Wro4J	No	No	No	Yes (3)	No	No	No	No	No
8	JGAP	No	No	No	Yes (2)	No	No	No	No	No
9	ArgoUML	No	Yes (1)	Yes (12)	Yes (30)	No	No	Yes (2)	Yes (1)	No
10	Mars	No	Yes (2)	No	Yes (2)	No	No	No	Yes (1)	No
	Total no. of diagrams used (i.e. no. of '*yes*')	**1**	**2**	**2**	**10**	**0**	**0**	**2**	**2**	**0**
	Total no. of Diagram	**1**	**3**	**13**	**55**	**0**	**0**	**3**	**2**	**0**

used. The term 'Yes' in Table 3.3 means that the project used at least one instance of a UML diagram specified in the table. The numbers in Table 3.3 indicate the number of diagrams constructed in the OSSD projects. Similar to most industrial use, none of the OSSD projects used UML to model their complete system. 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 showed 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.

Use Case Diagram

A use case diagram is used to describe the desired functionality of the software product [65]. Use case diagrams were used by only one of our evaluated OSSD project (see JPMC in Table 3.3). Most of these OSSD projects have specified their goal, but the specification and the interaction between users and system were explained in text.

Component Diagram

Component diagrams are used to divide the system into components and show their relationships through breakdown of components into lower-level structure [63]. This diagram is used to illustrate the high-level structure of large systems. Because of this

reason, only complex projects among the case studies used this diagram. ArgoUML and the Mars Simulation Project provided this diagram in their repositories. 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. 'Top Level Diagram' and 'Simulation Component Diagram'. The 'Top Level Diagram' illustrated dependencies between 3 components while the '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 [63]. Package diagrams show the relationship between higher level units. This diagram is used to explain the high-level structure of a system. Only two of the case studies used this diagram. 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 domain-related packages and two other packages representing external libraries. The second package diagram illustrated the high-level package in this project.

Class Diagram

Class diagram is the most used UML diagram in these case studies. Most of the case studies only show classes that are important to the system. The correspondence between design classes and implementation is discussed in subsection 3.5.2.

Sequence/Interaction Diagram

Sequence diagrams were used by two of our evaluated OSSD projects. However, both projects have only one sequence diagram per project. ArgoUML introduced a sequence diagram after eight version of releases. Table 3.7 shows that only after version 0.26, a sequence diagram was introduced in the documentation. Perhaps, it is difficult to generate the sequence diagram for the entire release. Hence, the developer of this project used a sequence diagram to illustrate the flow of a new feature. The gwt-portlets project used only one sequence diagram. We assume that the described flow contains crucial information for the system. This is due to the classes that were involved in the sequence diagram were presented in the project's 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 behaviour [68]. This study found that two OSSD projects used the activity diagram. However, not all activity diagrams in these projects were related to the software development. ArgoUML used an activity diagram to present the flow of managing issues in ArgoUML project. Meanwhile, the Mars Simulation project used one activity diagram for specifying a feature of the project.

3.5.2 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. 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 the implementation. This is a relatively small project that 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 was found in the JavaClient project with 57 classes. The data in Table 3.4 is depicted graphically in Figure 3.1. This figure shows that the ratio between the number of classes in the design and the number of classes in the implementation decreases when the number of classes in the implementation increases. Based on our observation, most of the projects had created UML diagrams in the early version of the release, but rarely increase the amount of classes in the design.

3.5.3 Level of Detail (LoD)

This subsection describes the result of assessing the level of detail used in modeling, as well as the use of reverse engineering in the case studies. The overall result is illustrated in Table 3.5. The project that uses the most class diagrams is ArgoUML: 46 class diagrams. Out of this total number, 19 UML diagrams were constructed in Low LoD and other 27 diagrams were constructed in High LoD. ArgoUML is the only case study that used RE-CDs. There were 15 diagrams which were constructed using reverse engineering techniques. Almost half of these diagrams were used to describe the user interface from an old design documentation and other diagrams showed class diagrams for selected classes. Most of the selected classes were classes that play a key role in how the program works [165]. The Maze project showed some interesting result in their construction of UML diagrams. There were six UML class diagrams constructed in this project. A class diagram with low LoD displays 40% of the total classes. These diagrams illustrate the relationships between domain-related classes

Table 3.4: *Classes in Design versus Classes in Implementation*

No	Project	# of Class Diagram	# of Classes in Design (D)	# of Classes in source code (S)	% D vs S
1	Maze	6	28	69	40.6
2	JavaClient	1	57	215	26.5
3	xUML	1	45	172	26.2
4	JPMC	4	24	126	19.1
5	Neuroph	3	24	179	13.4
6	gwt-Portlets	3	20	178	11.2
7	Wro4J	3	11	100	11.0
8	JGAP	2	18	191	9.4
9	ArgoUML	30	33	909	3.6
10	Mars Simulation	2	31	953	3.3
	Total	**55**	**291**	**3092**	**16.4**

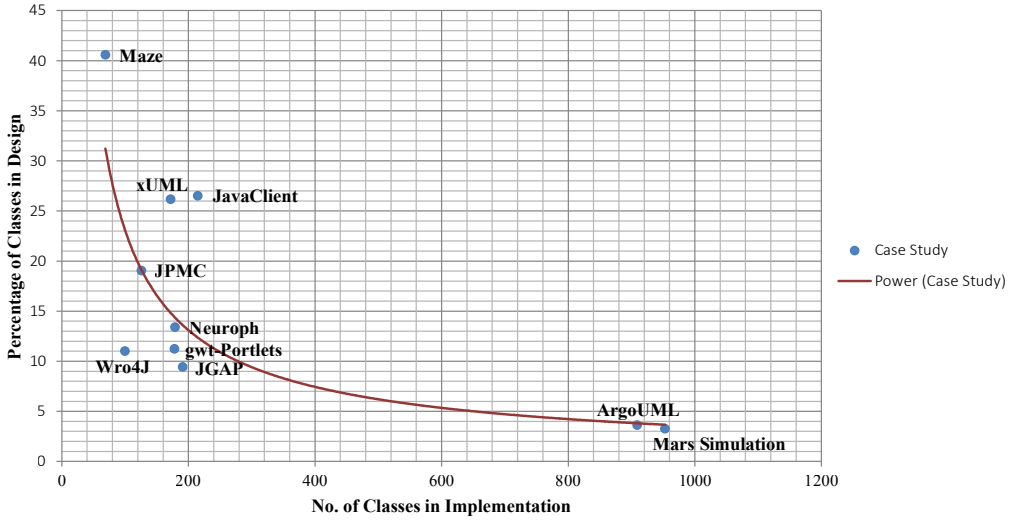

Figure 3.1: *Classes in Design vs Classes in Implementation*

Table 3.5: *LoD and Forward/Reverse Class Diagram*

No	Project	Low LoD	High LoD	Forward	Reverse	Total Class Diagram
1	**ArgoUML**	16	14	15	15	30
2	**Maze**	1	5	6	0	6
3	**JPMC**	0	4	4	0	4
4	**Mars**	2	0	2	0	2
5	**Wro4J**	2	1	3	0	3
6	**Neuroph**	1	2	3	0	3
7	**Gwt-Portlets**	1	2	3	0	3
8	**JGAP**	2	0	2	0	2
9	**Javaclient**	0	1	1	0	1
10	**xUML**	0	1	1	0	1
	Total	25	30	40	15	55

and external library classes used in this system. The other five diagrams presented class diagrams based on selected packages. We assume that the classes listed in the design play an important role in the system. In the Mars Simulation project, the activity diagram and the component diagram were constructed with in LoD.

The Wro4J project used only class diagrams. Two of the class diagrams were in Low LoD, while another class diagram was constructed in High LoD. This Low LoD class diagram was used to describe the structure of classes in the system. Similar to this project, Gwt-Portlets project also used a Low LoD class diagram to present the high-level class structure in the project. Another two class diagrams showed the classes and the relationship of important classes in the system. The Low LoD was also used to present the higher level of abstraction of the system class diagram in JGAP project. This project has two class diagrams from different versions of releases that showed the class diagrams of the key classes in the system.

In Neuroph, three class diagrams were presented in the system repository. Two of the diagrams were presented in High LoD. Those class diagrams were used to describe the class diagrams of important or key classes in the system. Meanwhile, all JPMC project's UML diagrams were constructed in High LoD. There were four class diagrams showing the key classes of the system. The high-level abstractions of class structures were presented in a package diagram and a component diagram.

The JavaClient and xUML-compiler project have only one High LoD diagram for each project. The JavaClient project constructed a very complex and high LoD class diagram that consisted of 67 classes. This class diagram consisted of classes that existed in the first version of this system. Hence, it is different from other case studies that normally showed the important, relevant or key classes in the system. The xUML-compiler project constructed a High LoD class diagram that show the relations between the domain-related classes and the external packages.

In addition to LoD, we also evaluate the usage of reverse engineering in OSSD projects. Initially, we expected the class diagrams that were constructed using reverse engineering to have a High LoD. However, we found several RE-CDs that had Low LoD in ArgoUML project. We found that only ArgoUML used reverse engineering for reconstructing some of its class diagrams. However, several UML diagrams from other projects also show several symptoms of RE-CDs (for example, "no aggregation and composition relationship" and "multiple relationships for the same direction"). Perhaps, these diagrams were constructed through reverse engineering, but were subsequently modified manually and ended up looking like a forward engineering design.

3.5.4 Frequency of Updating UML Models

This subsection presents the frequency of updating the UML models of the case studies. Basically, we would like to know whether UML diagrams are used throughout the projects or only in the 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 on Dec 2009. Based on that date, we assume that this design corresponds with the 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 diagrams is low. In most of the case studies, a new UML diagram was created when there was a new feature of the system introduced in a new version or release. Only the Neuroph and ArgoUML project actually modified existing diagrams. Other projects 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 increased. The work by Wen Zhang et al. [186] shows that there was an increasing amount of participants in version 0.26. As we can see in Table 3.6, 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 newcomer developers that was looking for information about the design. The creation of UML diagrams is perhaps being used to ease the communication of the new developers. It is also possible that the new software developers created this diagram to help their understanding of the system. In the next subsection, we discuss the ArgoUML project as an example of a project that did update their UML designs across multiple versions of releases.

Table 3.6: *Add, Remove and Modify of UML Diagrams in ArgoUML Project*

No	Release Version	UML Diagram			Remarks
		Add	Remove	Modify	
1	0.10.1	15	0	0	Old Document
2	0.12	18	0	0	Cookbook 2003 was added
3	0.14	0	0	0	Cookbook 0.14 was added but no changes for UML diagram
4	0.16	1	1	0	Cook book 0.16 was added. Key classes class diagram was taken out
5	0.18.1	0	0	0	Cookbook 0.18.1 was added but no changes for UML diagram
6	0.20	3	0	0	Cookbook 0.20 was added.
7	0.22	0	0	0	Cookbook 0.22 was added but no changes for UML diagram
8	0.24	1	0	0	Cookbook 0.24 was added
9	0.26	8	0	3	Cookbook 0.26 was added.
10	0.26.2	0	0	0	Cookbook 0.26.2 was added but no changes for UML diagram

ArgoUML

Table 3.7 shows which types of diagrams were used across subsequent versions over time. The table shows that in the early versions of the software, 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 (for non-embedded applications): Firstly, the developers design the overall structure (using component diagrams) and later continue to flesh out using behavioural diagrams of the design. Figure 3.2 shows the evolution of UML diagrams in every version of release. Figure 3.2 also shows the evolution of the number of classes. It is explicitly shown that UML diagrams are rapidly created in the early stage of software release and then occasionally updated.

3.5.5 Key Classes

This study shows that UML diagrams do not cover the entire scope of the implementation. Class diagrams only show the key classes in the system. The OSSD developers identified the main classes of the system and showed these in a high-level class diagram. Perhaps, the package diagram is usually not used to present the high-level abstraction because it is too brief and a complete class diagram is not used because it contains too much information. In addition, we contacted one of the developers of the case studies. He confirmed that the developer in his project constructed UML diagrams only for important classes. Thus, we believed that there is a need for a class diagram abstraction or condensation method to produce this kind of diagram; as also suggested by Ichii et al. [82]. The studies performed by Andriyevska et al. [17] and Osman et al. [133] may be useful for the class diagram abstraction.

Table 3.7: *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

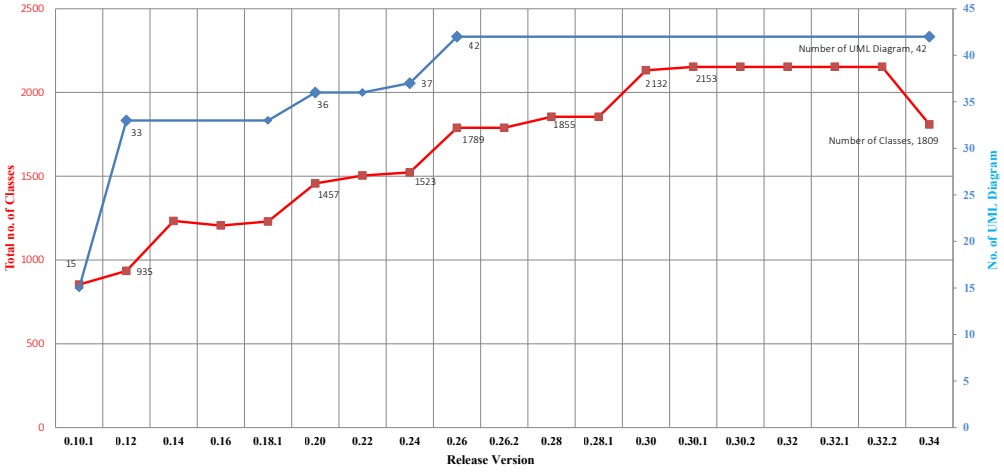

Figure 3.2: *ArgoUML Evolution in UML Diagrams and Number of Classes*

3.5.6 Threats to Validity

This section describes the threats to validity of this study. In terms of case study selection, there could be more case studies if we include more open source repositories and also include projects developed other than Java programming language. The selected projects may not be representative of all the OSSD because the selected case studies can be considered as small and medium type of system development and also specific to Java-based project. In addition, referring to figure 3.1, we also do not have projects with a number of classes between 250 and 800. The result could be different if more large projects are included in this study. We used the keywords of "Class Diagram" when we search for the suitable case study. We realized that there are possibility that we missed some projects that have UML diagrams, but stored in CASE tool's format (such as .zargo-ArgoUML and .uml-StarUML). The study was done based on using only 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 uses 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.

3.6 Conclusion and Future Work

This study explored if UML diagrams are used in OSSD projects. To this end, ten case studies were collected from online repositories. Four main questions were studied: What types of UML diagrams are used? How does the ratio of the design relates to the size of the implementation? What level of detail is used in UML diagrams? and How does timing of changes in the implementation relate to the changes in UML diagrams/documentation? The main purpose of UML modeling in projects is to ease the communication between the developers. This also seems to apply to OSSD projects. UML diagrams (specifically class diagrams) with a low level of detail are used to show a high-level abstraction of the structure of the system. UML diagrams with a high-level of detail are used to elaborate key aspects of the design or complex aspects of the design. 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 chapter shows that open source projects can be used as empirical sources for studying the 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 for their pattern in updating UML models. Furthermore, future work could be to find more case studies and to extend the case studies to languages other than Java. This would allow differentiating results between programming languages.