UML Class Diagram Simplification
A Survey for Improving Reverse Engineered Class Diagram Comprehension

Hafeez Osman[1], Arjan van Zadelhoff[1] and Michel R. V. Chaudron[1,2]
[1]*Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands*
[2]*Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden*

Keywords: Software Engineering, Reverse Engineering, UML, Class Diagram, Simplification.

Abstract: Class diagrams may include an overwhelming amount of information. For a large and complex class diagrams there is a possibility that not all information in the class diagram is important for understanding the system. In this paper, we study how to identify essential and secondary information in class diagrams. To this end, we performed a survey with professionals, academics and students to enquire information in class diagrams that is considered important. In total, 25 complete responses were received with 76% of the respondents having average or above skills with class diagrams. As the results, we discovered that the metric that counts the number of public operations is the most important metric for indicating importance of a class in a diagram. Also, we discovered that class names and coupling were influencing factors when it comes to excluding classes from a class diagram.

1 INTRODUCTION

UML class diagrams are valuable artefacts in software development and software maintenance. This diagram is helpful for software developers and software maintainers in order to understand architecture, design and implementation of a software system. UML class diagrams describe the static structure of programs at a higher level of abstraction than source code (Guéhéneuc, 2004). UML models, which are usually created during the design phase, are often poorly kept up-to-date during the realization and maintenance phase. As the implementation evolves, correspondence between design and implementation degrades from its initial design (Nugroho and Chaudron, 2007).

Reverse engineering is the process of analyzing the source code of a system to identify its components and their interrelationships and create design representations of the system at a higher level of abstraction (Chikofsky and Cross, 1990). In particular, reverse engineered class diagrams are typically a detailed representation of the underlying source code, which makes it hard for the software engineer to understand what are the key elements in the software structure (Osman and Chaudron, 2011). Although several Computer Aided Software Engineering (CASE) tools have options to leave out several properties in a class diagram, they are unable to automatically identify important classes.

Normally, to understand a software system, a programmer needs both source code and design. A good representation of a class diagram by showing the crucial information of a system is needed, especially when new programmers want to join a development group; they need a starting point in order to understand the whole project before they are able to contribute. Tools that support maintenance, re-engineering or re-architecting activities have become important to decrease the time software personnel spend on manual source code analysis and help to focus attention on important program understanding issues (Bellay and Gall, 1997). This paper aims at simplifying UML class diagrams by leaving out unnecessary information without affecting the developer's understanding of the entire system. To this end, we have conducted a survey to gather information from professionals, academics and students about what type of information they focus on.

The paper is structured as follows : Section 2 discusses related work. Section 3 describes the examined properties and tools for this research. Section 4 explains about the survey methodology while Section 5 presents the results and findings. We discuss our findings in Section 6 and Section 7 suggests future work. This is followed by our conclusion in Section 8.

Osman H., van Zadelhoff A. and R. V. Chaudron M..
UML Class Diagram Simplification - A Survey for Improving Reverse Engineered Class Diagram Comprehension.
DOI: 10.5220/0004319902910296
In *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2013), pages 291-296
ISBN: 978-989-8565-42-6

2 RELATED WORK

In this section, we discuss several studies that are slightly related to this study. Yusuf et al. (Yusuf et al., 2007) performed a study about assessing the comprehension of UML Class diagrams via eye tracking. They used eye-tracking equipment to collect a subject's activity data in a non-obtrusive way as the subjects are interacting with the class diagram in performing a given task. They concluded that experts tend to use such things as stereotype information, colouring, and layout to facilitate more efficient exploration and navigation of class diagrams. Also, experts tend to navigate/explore from the center of the diagram to the edges whereas novices tend to navigate/explore from top-to-bottom and left-to-right.

Egyed (Egyed, 2002) presents an approach for automated abstraction that allows designers to zoom out of class diagrams to investigate and reason about the bigger picture. This approach was based on a large number of abstraction rules and, when used together, it can abstract complex class structures quickly. In total, the article provides 121 rules to abstract a class diagram. They showed that their technique scales, produces correct results most of the time, and addresses issues such as model ambiguities that are inherently part of many (UML) diagrams.

Table 1: The Chosen Software Design Metrics.

No	Metrics	Category	Description
1.	NumAttr	Size	The number of attributes in a class
2.	NumOps	Size	The number of operations in a class
3.	NumPubOps	Size	The number of public operations in a class
4.	Setters	Size	The number of operations with a name starting with 'set'.
5.	Getters	Size	The number of operations with a name starting with 'get', 'is', or 'has'.
6.	NOC	Inheritance	The number of immediate subclasses subordinated to a class in the class hierarchy.
7.	DIT	Inheritance	DIT is calculated as the longest path from the class to the root of the inheritance tree.
8.	CLD	Inheritance	The longest path from the class to a leaf node in the inheritance hierarchy below the class.
9.	Dep_Out	Coupling (import)	The number of dependencies where the class is the client
10.	Dep_In	Coupling (export)	The number of dependencies where the class is the supplier
11.	EC_Attr	Coupling (export)	The number of times the class is externally used at attribute type
12.	IC_Attr	Coupling (import)	The number of attributes in the class having another class or interface as their type
13.	EC_Par	Coupling (export)	The number of times the class is externally used as parameter type
14.	IC_Par	Coupling (import)	The number of parameters in the class having another class or interface as their type

3 EXAMINED PROPERTIES AND TOOLS

In this section we describe the design metrics and the tools used for this research.

3.1 Examined Properties

We study 14 metrics from the categories Size, Coupling and Inheritance. The metrics that we use are listed in Table 1.

3.2 Tools

SDMetrics[1] is used to measure the structural properties of object oriented design. SDMetrics version 2.11 (academic license) is used for this purpose. We chose Enterprise Architect[2] for design and reverse engineered UML class diagrams in this survey.

4 SURVEY METHODOLOGY

In this section, we explain how the questionnaire was designed and why.

4.1 Questionnaire Design

The questionnaire consisted out of 3 parts. A printable version is available at (van Zadelhoff, 2012).

4.1.1 Part A: Background of the Respondents

Part A consisted of 4 questions. We asked about the respondent's background: occupation, location, years of experience with class diagrams and skills in creating, modifying and understanding class diagrams.

4.1.2 Part B: Class Diagram Indicators for Class Inclusion /Exclusion

The first 13 questions asked which metrics of a class diagram based on indicator for including/excluding a class. In each of these 13 questions, we briefly explained the metrics and a choise of 5 multiple choises were offered are shown in Table 2. In the last question of part B (i.e. question 14), we tried to discover the reasons of the respondents for including and excluding a class in a class diagram.

Table 2: Answers Multiple Choice Questions.

Multiple Choice Letter	Answers	Score
A	Class(es) **Definitely Should Not** be Included	-2
B	Class(es) **Probably Should Not** be Included	-1
C	Class(es) **Sometimes** be Included	0
D	Class(es) **Probably Should** be Included	1
E	Class(es) **Definitely Should** be Included	2

[1] http://www.sdmetrics.com/

[2] http://www.sparxsystems.com.au

4.1.3 Part C: Commenting UML Diagrams

In this part, we tried to simulate actual problem by providing several class diagrams. Participants had to indicate which class or information of a class to include/exclude. The following class diagrams were involved in this survey:

1. **ATM simulation system:** This class diagram is developed by the Department of Mathematics and Computer Science, Gordon College (Bjork, 2004).
2. **Library System:** This system is taken from (Eriksson et al., 2004). The reverse engineered design was used for this questionnaire.
3. **Pacman Game:** This project is found at (Craig et al., 2009). We used the forward design and the reverse engineered design of this system

We also tried to simulate the various flavours of class diagrams from the software industry by providing different Levels of Detail (LoD) of class diagrams and the sources of class diagrams. The information about the class diagrams that we used in question number 1, 2, 3 and 5 in part C is shown in Table 3. Other than these 4 questions, we have made 2 more questions in which we asked the respondents which flavour of class diagram they prefer.

Table 3: Description of Design Models used in the Questions.

Question	System	Source of Diagram	Level of Detail (LoD)
1	ATM Machine	Forward Design	Low
2	Library System	Reverse Engineered	High
3	Pacman Game	Forward Design	High
5	Pacman Game	Reverse Engineered	High

4.2 Experiment Description

The questionnaire was published online for two months. The total respondents that started this questionnaire were 98. However, only 25 respondents completed this questionnaire. Most of the incomplete responses stopped after the questions in Part A.

5 RESULTS AND FINDINGS

In this section we present our results and findings from this survey. The responses for this survey are available at (Osman and van Zadelhoff, 2012).

5.1 Background of the Respondents

In this subsection we present the results of part A of the questionnaire.

5.1.1 Roles and Locations

For the respondents' role, 40% of the respondents mentioned that their current status is Researcher/Academic while 32% of the respondents are IT Professionals. 28% of the respondents answered Student. For the respondents' location, 45% of the respondents stated that they were from in The Netherlands. 32% of the respondents are from Malaysia.

5.1.2 Skills and Experience with Class Diagrams

24% of the respondents mentioned that their experience with class diagrams is between 1 and 3 years while 16% of the respondents answered this question with "3 - 7 Years". 12% of the respondents answered "7 - 10 Years" and 20% of the respondents mentioned that they have more than 10 years of experience with class diagrams. In terms of the respondents skills on creating, modifying and understanding class diagrams. 40% of the respondents stated that their skill is Average, while 20% answered "Good". 16% of the respondents rated their skill Excellent.

5.2 Indicator for Class Inclusion

In this subsection we present our result of part B. For question B1 to B13, the respondents were provided 5 choises of answers. We analysed these 13 questions by using a score-system as shown in Table 2.

In the Size category, the highest score is NumPubOps with 25 points. NumOps has 18 points while NumAttr has 17 points. The Setters/Getters metric has 13 points. From these results we can consider that the respondents found operations are important in a class diagram, specifically operations that are public.

For the Coupling category, Dep_Out and Dep_In have 17 and 16 points, respectively. EC_Attr has 15 points while IC_Attr has 17 points. If we compare the points between these two metrics and EC_Par (11 points) and IC_Par (9 points), there is a huge difference. This indicates that the classes that are declared and are used as an attribute are more important than the classes that are declared and are used as a parameter in class operations.

For the Inheritance category, the results indicate that NOC has the most points in this category which is 20 points. DIT and CLD have 7 points and 5 points, respectively. With these results we can suggest that

the respondents only need a part of the inheritance tree.

In term of class inclusion/exclusion, the answers are analyzed by creating several keywords that are related to the given answers An answer in this open-ended question could contain multiple keywords. We understand that the "Important/Relevant class" is a very broad term but that is basically what the respondents answered. The answers also show that there are three keywords that are being related to the answers the most. These are Important/Relevant Class (29.6%), Coupling (18.5%), and Domain Related (25.9%).

5.3 Practical Problems

In this part, we tried to access the information about the classes that should not be included in a class diagram.

5.3.1 Coupling

In question C1, the ATM System class diagram was presented without attributes and operations. As a result, 48% of the respondents chose to exclude the class Money and 36% of the respondents chose to not include the OperatorPanel and Status class in a class diagram. From our observation, those 3 classes have the number of coupling (e.g. Dep_In,Dep _Out) $<=2$. 32% of the respondents chose to exclude the classes Deposit, EnvelopeAcceptor, ReceiptPrinter, Transfer and Withdrawal. The coupling for those classes is $=2$. This shows that the amount of coupling plays a major role in selecting the classes that should or should not be included in a class diagram.

5.3.2 Meaningful Class Names

A reverse engineered class diagram from a Library System was used for question C2 an all elements were presented in High Level of Detail (HLoD). We were expected to discover the elements that influence in selecting the classes that should not be included.

From the results, it is shown that most of the respondents chose not to include the classes that have no relationship. In this question, we found that class names also play a major role in determining whether a class should be included or excluded. AboutDialog, MessageBox and QuitDialog clearly mentioned the functionality of the classes that are used to display the information. Borrower, Reservation, Loan, Item and Title are classes that have a meaningful name that might indicate the functionality of the classes and also closely relate to the domain i.e. Library System.

5.3.3 Level of Detail

In Question C4, by referring to the class diagrams in C1, C2 and C3, the respondents were asked which flavour of class diagram they preferred.

The results show that almost half of the respondents preferred working with class diagram C3. 48% of the respondents preferred diagram C3 because they mentioned that the class diagram is clear, the necessary information is provided e.g. attributes and operations and most of the classes that are presented are important. 20% of the respondents preferred to use class diagram C1. Most of the respondents that chose this diagram were Researchers/Academic and IT Professionals with the skill in class diagram ranging from Average to Excellent. It seems that most of the respondents that have a good skill and experience in class diagrams prefer to use this diagram. 12% of the respondents mentioned that it did not matter which diagram they get, 8% of the respondents preferred class diagram C2 and 8% did not prefer any of the presented class diagrams.

5.3.4 Reverse Engineered Class Diagram which Conformed to Forward Design

In question C5, the class diagram used was slightly different from the class diagram presented in question C3 because this class diagram was constructed by using a reverse engineering technique. In this question, we tried to discover if there was any difference of selecting the classes that should not be included in a class diagram in a reverse engineered class diagram that is close or almost similar with the forward design class.

The result shows that the class Direction and PacShell were selected by 72% of the respondents to be left out from the class diagram. Compared to the question C3, the Iterator and Iterable classes were differently presented in this reverse engineered diagram. The interface class is automatically presented in the class that is connected to the interface class.

5.3.5 Reverse Engineered vs. Forward Engineer Class Diagram

Question C6 tried to discover which type of class diagram was preferred by the respondents i.e. between the reverse engineered and the forward engineered class diagrams. The reverse engineered class diagram used was different with the reverse engineered class diagram in question C2 because this class diagram was derived from a system that was implemented (or coded) closely with the forward design.

The results show that most of the respondents (mainly researcher) preferred to use the reverse engineered class diagram (Class Diagram C5). 40% of the respondents chose this diagram because it is more detailed, clear, there is no interface class and it is easier to understand. 20% of the respondents did not choose any of the two class diagrams because for them it does not matter which one. On the other hand, 16% of the respondents preferred forward engineered class diagram (class diagram C3).

6 DISCUSSION

In this section we discuss the results and findings presented in the previous section.

6.1 Respondents' Background

In Part A, the respondent's status in this questionnaire was quite evenly distributed and the location of the respondents showed that most of the respondents are from The Netherlands and Malaysia.

In terms of the respondent's skill and experience with class diagrams, we found that 72% of the respondents have more than 1 year of experience and that 76% of the respondents have rated themselves average or above if it comes to creating, modifying and understanding class diagrams. Even though 28% of the respondents said that they have less than one year of experience, we can still state that all the respondents have knowledge about class diagrams.

6.2 Software Design Metrics

In the Size category, we found that the higher the number of public operation, is the more people prefer this class. Public operations are not restricted to be accessed internally but they also can be accessed publicly from other classes. This might be the reason why the respondents found public operations.

In the Coupling category, we have discovered that classes that have many incoming and outgoing dependencies are important. We found that IC_Attr and EC_Attr have higher points than EC_Par and IC_Par. The reason might be that the class(that is declared as an attribute) is more important because it could be used for every operation in the class. In the Inheritance category, we discovered that for a class that has a high NOC, the class should be included in a class diagram. This parent class is helpful to show the abstraction of a group of classes. For DIT, the higher number of DIT does not indicate it is an important class because it basically means that this particular class is located very low in the inheritance hierarchy which means that this class is too detailed and most of the times not needed. For CLD, if a class has a high frequency of this metric means this class is very abstract, meaning that this class alone will not be enough to understand the whole hierarchy.

As for the complete results, we found that NumPubOps has the highest points of all the metrics. Also all the metrics have a positive score. The overall ranking of the score is shown in Table 4. This result could be applied for a software designer to simplify a class diagram during the documentation phase.

Table 4: Overall Score for Software Design Metrics.

No.	Software Metrics	Score
1	NumPubOps	25
2	NOC	20
3	NumOps	18
4	NumAttr	17
5	Dep_Out	17
6	IC_Attr	17
7	Dep_In	16
8	EC_Attr	15
9	Setters/Getters	13
10	EC_Par	11
11	IC_Par	9
12	DIT	7
13	CLD	5

6.3 Class Names and Coupling

Based on the results in Part C, we discovered that a highly influential factor when we are trying to exclude classes from a class diagram is coupling. Another influencing factor is the class name. Many respondents excluded Graphical User Interface (GUI) related classes in the Library system because of the class name and coupling. However, sometimes this element is not an influencing factor as we have seen in the ATM system because many respondents excluded domain related classes, classes that are needed for the functionality of the ATM system.

6.4 Class Diagram Preferences

In question C4, we discovered that most of the respondents preferred to use HLoD of the forward design. The reasons the respondents gave was that this class diagram is clearer and the necessary information is provided in this class diagram. This result seems to indicate that the forward design with High Level of Detail was preferred by the respondents. Meanwhile, in question C6, most of the respondents had chosen the reverse engineered design (HLoD). The reason

might be that the reverse engineered design that was provided has few differences from the forward design. The respondent stated that they preferred this diagram because they find it more detailed, clear, and it is easier to understand. Some of the respondents also mentioned that the interface classes are removed and is thus a better class diagram.

6.5 Threats to Validity

Although the respondents of this survey was quite well distributed between the status roles (Student, Researcher/Academic and IT Professional), we consider that the amount of full responses were not enough. The locations of the respondents were biased to The Netherlands and Malaysia. Most of the questions in this study require the respondent to choose the best answers. We needed to do several assumptions on why the respondents chose these answers and this assumptions may not be accurate.

7 FUTURE WORK

This study was an early experiment on how to simplify class diagrams and we see a number of ways to extend this work. We propose to validate the resulting class diagram by using an industrial case study and discover the suitability of the simplified class diagram for the practical usage. It would also be interesting to include other metrics that we have not chosen and check whether they are important or not and ask why the respondent chose the answer to get the reason.

From the results, we found that class role and responsibility are one of the important indicators in a class diagram. We would like to suggest a study on names (class, operation and attribute) that the software developers find important or meaningful in order to understand a system. We discovered some weakness in the questionnaire and our suggestion is to improve this questionnaire by increasing the amount of responses. It would be interesting to see what the results are with a larger group of respondents.

8 CONCLUSIONS

In this survey we have discovered the most important elements in a class design. We also discovered what flavour of class diagrams is preferred to work with. From the results, we discovered that the most important software design metric is the Number of Public Operations. This means that if a class has a high number of public operations then this indicates that this class is important and should be included in a class diagram. In this survey we also discovered that the class names and coupling are influencing factors when selecting a class to be excluded from a class diagram.

With these results we can highlight which classes should be included or excluded in a reverse engineered class diagrams based on our results and analysis by looking at the metrics and behaviour the respondents had in Part C. Although the number of responses of this questionnaire is not that high, we still managed to find some influencing factors for deciding a class to be included or excluded in a class diagram.

REFERENCES

Bellay, B. and Gall, H. (1997). *A Comparison of Four Reverse Engineering Tools*, pages 2–11. IEEE Computer Society Press.

Bjork, R. C. (2004). Atm system. http://www.math-cs.gordon.edu/courses/cs211/ATMExample/ .

Chikofsky, E. J. and Cross, J. H. (1990). Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17.

Craig, A., Dinardo, A., and Gillespie, R. (2009). Pacman game. http://code.google.com/p/tb-pacman/ .

Egyed, A. (2002). Automated abstraction of class diagrams. *ACM Trans. Softw. Eng. Methodol*, 11(4):449–491.

Eriksson, H.-E., Penker, M., Lyons, B., and Fado, D. (2004). *UML 2 Toolkit*. Wiley.

Guéhéneuc, Y.-G. (2004). *A Systematic Study of UML Class Diagram Constituents for their Abstract and Precise Recovery*, pages 265–274. IEEE.

Nugroho, A. and Chaudron, M. R. V. (September 20-21, 2007). *A Survey of the Practice of Design - Code Correspondence amongst Professional Software Engineers*, pages 467–469. Proceedings of the First International Symposium on Empirical Software Engineering and Measurement.

Osman, H. and Chaudron, M. R. V. (September 12-13, 2011). *An Assessment of Reverse Engineering Capabilities of UML CASE Tools*, pages 7–12. 2nd Annual International Conference Proceedings on Software Engineering Application (SEA 2011).

Osman, H. and van Zadelhoff, A. (2012). Structured questionnaire responses. http://www.liacs.nl/∼hosman/Complete_Results_Structural_Survey.rar.

van Zadelhoff, A. (2012). Structured questionnaire. http://www.liacs.nl/∼hosman/The_Presence_of_Classes_in_Class_Diagrams.pdf.

Yusuf, S., Kagdi, H., and Maletic, J. I. (2007). *Assessing the Comprehension of UML Class Diagrams via Eye Tracking.* pages 113–122. 15th IEEE International Conference on Program Comprehension ICPC '07.