Internal Report 2012-05 August 2012

Universiteit Leiden

Opleiding Informatica

UML Class Diagram Simplification: A Survey Study

Arjan van Zadelhoff

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Contents

UML Class Diagram Simplification: A Survey Study

By: Arjan van Zadelhoff
Supervisors: Hafeez Osman and Michel R. V. Chaudron

Leiden Institute of Advanced Computer Science
Leiden University, The Netherlands

August 16, 2012

Abstract

Class diagrams play an important role in software development. However, in some cases, these diagrams contain a lot of information that makes it hard for software maintainers to use them to understand a system. To reduce the information in a class diagram, a method to simplify a class diagram is needed. This simplified class diagram is resulting from leaving out details that are not needed and to remain the important information. To this end, we did 2 surveys to enquire the information about what type of information they would include or exclude in order to simplify a class diagram. The first survey involved 32 software developers with 75 percent of the participants having more than 5 years of experience in class diagrams. The second survey involved 25 respondents that answered this survey online, with 76 percent that rated themselves average or above in their skills in creating, modifying and understanding class diagrams. As for the results, we found that the important elements in a class diagram are class relationship, meaningful class names and class properties. We also found that, in a simplified class diagram, GUI related information, private and protected operations, helper classes and library classes should be excluded. In this survey we also tried to discover what types of features are needed for class diagram simplification tools.

1 Introduction

In this chapter we present the contexts, motivations, and objectives of this study. We also explain our research methodology, contributions and outline of this thesis. After reading this chapter, the reader should have the knowledge of our aim and problem that we attempted to solve.

1.1 Problem Statement

The UML class diagram is one of the valuable artefacts in software development and software maintenance. This diagram is helpful for software developers and software maintainers in order to understand architecture, design, implementation and behavior of a software system. UML class diagrams describe the static structure of programs at a higher level of abstraction than source code [15].

It is widely known that 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 [18]. For legacy software, reliable designs are often no longer available, while those are considered valuable for maintaining such systems.

Reverse engineering is one of the possible techniques to discover a software design after the implementation phase. 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 [10]. With this technique, recovery of a class diagram can be done by using the source code. However, the resulting class diagrams from reverse engineering techniques sometimes suffer with too much detail and information. 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 the key elements in the software structure are [19]. 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 classes and information that are not useful or less important. As part of a recent study [14], Fernández-Sáez et al. found that developers experience more difficulties in finding information in reverse engineered diagrams than in forward designed diagrams and also find the level of detail in forward designed diagrams more appropriate than in reverse engineered diagrams. For this reason, information that is needed by developers or maintainers to be shown in a class diagram should be discovered. In order to produce a better design phase representation of a class diagram, we need to discover the information that is important and not important in a class diagram.

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 modify. It is important for a new programmer to be able to identify which classes, attributes and operations play a major role in the system. Tools that support during maintenance, re-engineering or re-architecting activities has become important to decrease the time software personnel spend on manual source code analysis and help to focus attention on important program understanding issues [7]. Also, a method to assist the software engineer to focus on the relevant information and to leave out unnecessary information of the design is needed.

1.2 Objective

This thesis paper specifically aims at simplifying a UML class diagram by leaving out unnecessary information without affecting the developer's understanding of the entire software. To this end, we have conducted 2 surveys to gather information from our respondents about what type of information should not be included in a class diagram.

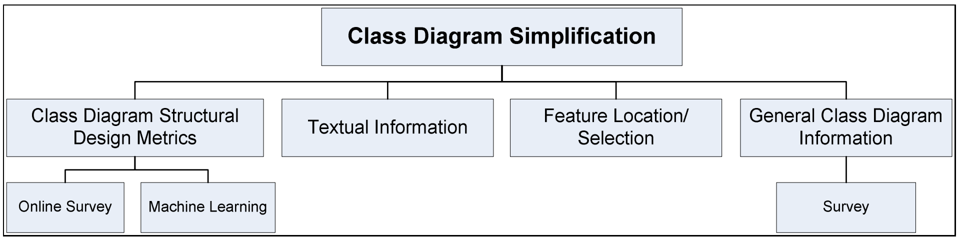
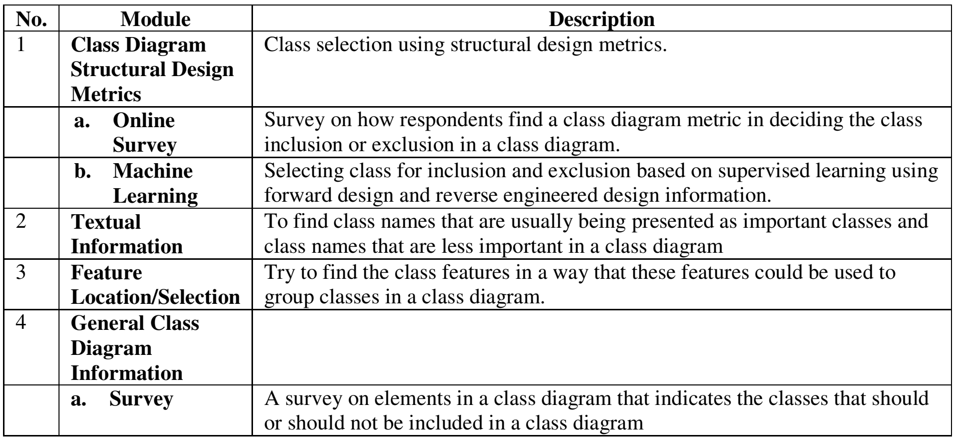

Class Diagram Simplification
Class Diagram Structural Design Metrics
Textual Information
Feature Location/ Selection
General Class Diagram Information
Online Survey
Machine Learning
Survey

Figure 1: Scope Class Diagram Simplification



No.	Module	Description
1	**Class Diagram Structural Design Metrics**	Class selection using structural design metrics.
	a. Online Survey	Survey on how respondents find a class diagram metric in deciding the class inclusion or exclusion in a class diagram.
	b. Machine Learning	Selecting class for inclusion and exclusion based on supervised learning using forward design and reverse engineered design information.
2	**Textual Information**	To find class names that are usually being presented as important classes and class names that are less important in a class diagram
3	**Feature Location/Selection**	Try to find the class features in a way that these features could be used to group classes in a class diagram.
4	**General Class Diagram Information**	
	a. Survey	A survey on elements in a class diagram that indicates the classes that should or should not be included in a class diagram

Table 1: Description Scope Terms

This study consisted of 2 surveys that were part of the Class Diagram Simplification study. The objectives between these two surveys were different. The overall study of Class Diagram simplification is illustrated in Figure 1 and Table 1 explains the terms. As shown in Figure 1, the first survey that we have done online was under the scope of Class Diagram Structural Design Metrics. This survey aimed at finding out which structural design metrics are important for class selection and class diagram simplification. This survey consisted of 24 questions which were divided into 3 parts in order to discover which metrics are important.

The second survey was under the scope of General Class Diagram Information. In this survey we tried to discover the elements in a class diagram that are needed to indicate whether a class should be or should not be included in a class diagram. This survey consisted of 15 questions that were also divided into three parts in order to discover the elements that the respondents find important in a class diagram.

1.3 Research Method

The research was a qualitative research [3] because we tried to discover the motives of the respondents in including or excluding a class in a class diagram. In other words, we tried to find out the reasons of this human behavior about why the respondents think a class should be excluded or included.

To discover this, we did a survey study which is a field research. We used this method because with a survey we could distribute it in many possible ways for example face-to-face, through email, using social media, or putting it on various forums. By using these options we could get a fair amount of responses in a short time. However, by distributing this survey, we had to be careful to which respondents we want to distribute because our target group is people that work with class diagrams or have knowledge about class diagrams and if this survey goes to some other group then the answers are not valid. To prevent this, we made some questions in which the respondent needs to state their background information, specifically their knowledge and experience in class diagrams.

1.4 Contributions and Outline

The contribution of this study is two-fold. First, the first survey highlights the information based on the class diagram metrics that are important to determine the class inclusion or class exclusion in a simplified class diagram. Second, the second survey emphasizes the other information about class inclusion and exclusion based on the software developer's evaluation and recommendation. This information can be used as an indicator to simplify class diagrams. It also could be basic information for the software developer/maintainer on how to determine important or relevant classes in a class diagram.

The organization of this study is as follows: Chapter 2 presents the background information. This chapter helps the reader to understand several things which are a prerequisite for reading this paper. In Chapter 3 we describe the first survey while chapter 4 describes the second survey. In these 2 chapters we give the survey methodology, results and finding, discussion and conclusion related to our findings. This follows with Chapter 5 in which we present the summary of the overall study, recommendations and future works. The chapter ends with the conclusions.

2 Background Information

This chapter describes the background information in which we explain a bit about the UML class diagram that we have used in this study. After that, we explain the different survey structures that we have used and what the differences between these two structures are. We then present the software design metrics that we have chosen for our online survey and we also explain why we have chosen these metrics. The tools we have used to assist the experiments are described after that. In the last part of this chapter we present some related works.

2.1 UML Class Diagram

The Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of object-oriented software engineering [5]. This standard is being managed and was created by the OMG (Object Management Group). UML has a set of different visual models to model an object-oriented system, such as sequence diagrams, use case diagrams, and so on. The current version of UML is now 2.4.1 and has now a total of 14 types of diagrams divided in 2 categories, which are structural and behavioral information. One of them, which we are using for our study, is the class diagram and is also in the structural category. In these structural diagrams the items that are being modeled must be presented in the system.

UML class diagrams describe the structure of a system. Such a diagram is shown in Figure 2. It describes the system by showing the classes and the relationships between the classes. In these classes, the attributes and operations are shown. Each attribute or operation can have a type such as integer, character, void and so on. The parameters of an operation can also be shown in this diagram. But these elements depend on the level of detail of the class diagram. There are many flavors of class diagrams used in the

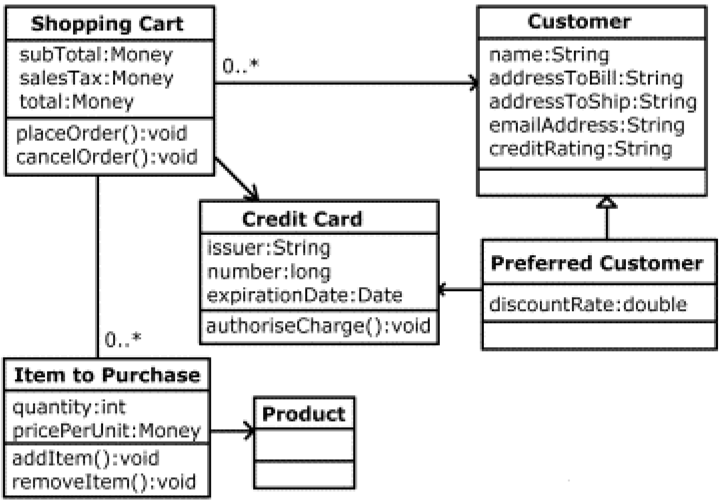


Figure 2: UML Class Diagram

software industry based on the Level of Detail (LoD). A class diagram may have only class names and their relationships between each other, which is a class diagram with a low level of detail (LLoD). A class diagram can also be presented with their attributes and their type, operations and their parameters, and the relations between the classes. This class diagram is shown in a high level of detail (HLoD). Different software developers/maintainers have their own preferences in the level of detail, which is most probably based on their experience and the given task for that particular system. These class diagrams are being modeled based on the user requirements. It is used to support the software developers and users to further understand the system.

2.2 Survey Structures

There are two types of questionnaires that have been applied during this project. The two types are:

- Structural Questionnaire
- Non-Structural Questionnaire

These two types are explained in the next subsections.

2.2.1 Structural Questionnaire

A structural questionnaire contains questions that offer the respondents a set of responses to choose from. This type of questionnaire is suitable for finding a result within a specific context. The process of collecting data is much simpler and the analysis of the answers takes less time to complete. Structured questions are best suited in the following situations: (1) the responses are understandable that the answer choices can develop; (2) it is not for capturing new ideas or thoughts from the respondent [24].

2.2.2 Non-Structural Questionnaire

A non-structural questionnaire or also known as open-ended questionnaire contains questions in which the answers are not offered for the respondents to choose from. This type of questionnaire is suitable for exploring new ideas. It is also helpful for discovering information that is unknown and discovering the expectations from the respondents. This open-ended questionnaire can be time consuming and difficult to analyze because the answers are unexpected and the range of the answers can be broad.

2.3 Software Design Metrics

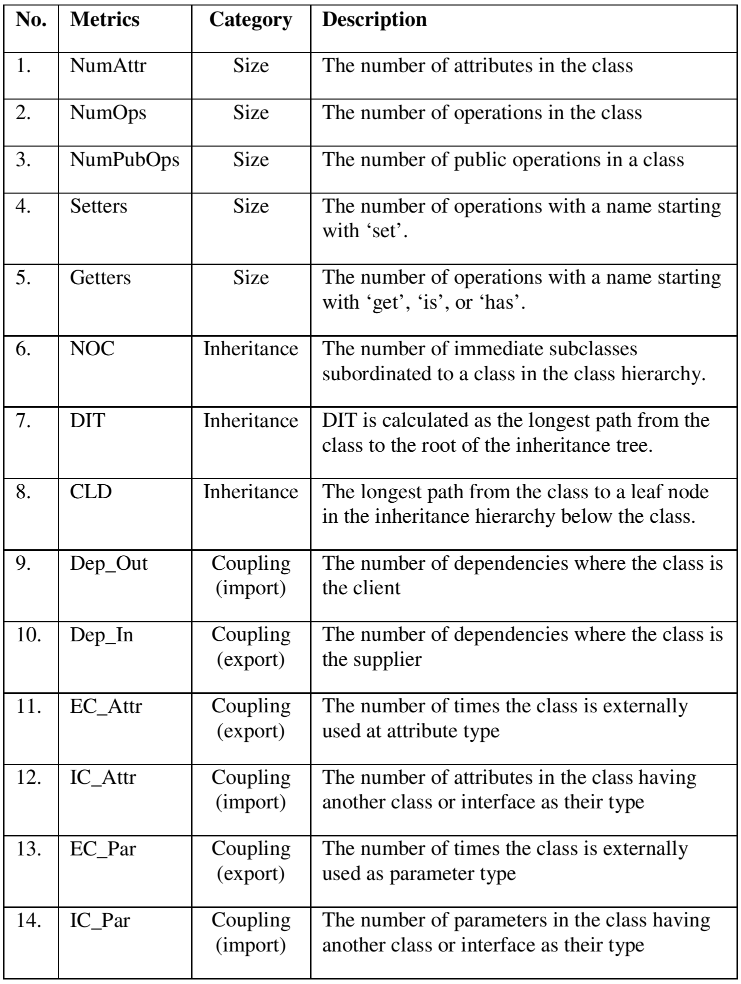
A design metric is an element in a class diagram that is being used for the structured questionnaire. For this survey we chose 14 metrics. These metrics were chosen from a program called SDMetrics [4]. SDMetrics counts how many times a certain metric is in a class diagram. A couple of class diagrams have been loaded in this program to compare which software design metric is suitable for this questionnaire. The chosen metrics are the metrics that have been occurred in the class diagrams the most (high numbers in SDMetrics). The chosen metrics are shown in Table 2.

2.4 Tools

In this section we briefly explain about the tools used for this study. The first part of this section describes the software metric tools that we have used in this study. In the next part of this section we explain about a simple statistical tool and the third part of this section describes the tools that we have used for designing and reverse engineering class diagrams.

2.4.1 Software Metric Tools

SDMetrics [4] is an object-oriented design measurement tool for the Unified Modeling Language (UML). SDMetrics is capable to measure 32 types of class diagram metrics which is divided into five (5) categories namely Size, Coupling, Inheritance, Complexity and Diagram. Nevertheless, only 14 metrics from category Size, Coupling and Inheritance are used in this study since the case studies only presented information in these software design metrics. The metrics that we have used are listed in Table 2.



No.	Metrics	Category	Description
1.	NumAttr	Size	The number of attributes in the class
2.	NumOps	Size	The number of operations in the 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

Table 2: The Chosen Software Design Metrics

2.4.2 Statistical Software

Microsoft Office Excel 2010 is used to table all the information gathered in both surveys. This tool offers simple features for statistical tasks such as calculation and diagram or graph construction.

2.4.3 Design and Reverse Engineering Tools

Enterprise Architect [1] is a modeling tool that offers various UML diagram design features, including class diagrams. This tool is being used for creating class diagrams for various questions in both questionnaires. This tool also offers reverse engineering features. By using these reverse engineering features, a UML class diagram can be reconstructed based on the source code. In both questionnaires, there are several reverse engineered models reconstructed by using this tool.

2.5 Related Works

To our knowledge no research is done in the way we approached our study. Some studies that have been done are slightly related. In this section we discuss those.

2.5.1 Eye Tracking

Yusuf et al. [25] did a study about assessing the comprehension of UML Class diagrams via eye tracking. They used eye-tracking equipment to implicitly 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. Also audio and video were recorded of every subject during these tasks. Their goal was to obtain an understanding of how human subjects use different types of information in UML class diagrams in performing their tasks. The tasks given to their subjects consist of the subjects answering specific questions by viewing the class diagrams. They created two types of questions: questions that deal with the basics of the class diagram and questions related to the software design. They concluded that experts tend to use such things as stereotype information, coloring, 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. Thus, subjects have a variation in the eye movements depending on their UML expertise and software-design ability to solve a given task.

2.5.2 Software Visualization

R. Koschke [16] did a study about software visualization in software maintenance, reverse engineering, and re-engineering. This study reports the results of a survey on the perspectives of 82 researchers in these 3 domains. The goal of this survey is to help to ascertain the current role of software visualization in software engineering from the perspective of researchers in these domains and give hints on future research avenues. This survey was sent by way of electronic mail to these researchers. The questions in this questionnaire were primarily open, so the respondents could answer freely what they want. Most of the questions were opinion-related questions, meaning that they ask the subject whether he/she thinks that visualization is appropriate (for example). Another part of the questionnaire was asking what kind of things they use to visualize software and how they visualize software. From this study it also seems that when they are visualizing artefacts, only 13 out of 82 subjects answered with UML, which is quite a lot.

The most answered with graph (49 subjects). The conclusion of this study was that many researchers of this survey prefer to only integrate the existing visualization techniques. Source code is also one of the most important artefacts for these subjects. Eventually, this survey has revealed a tendency to actually extend software visualization to what might be paraphrased as software perception.

Bassil et al. [6] did a survey study about software visualization (SV) tools that existed in 2000. This study addresses various functional, practical, cognitive and code analysis aspects that users may be looking for in SV tools. The participants, who are users of such tools in their industries or users that are in a research setting, rated the usefulness and importance of these aspects, and came up with their own desires. So basically, this questionnaire questions the SV tools on what has worked and what has not worked for these participants when applying a specific tool. The questionnaire is organized in two parts in which the first part is intended for any SV tool user, and the second part calls on expert users of SV tools. After the questionnaire, they analyzed it and in general, the participants were quite pleased with the SV tool at hand. Functional aspects such as searching and browsing, use of colors, and easy access from the symbol list to the corresponding source code were rated as the most essential aspects. Also hierarchical representations and navigation across hierarchies were strongly desired. Animation effects, 3D visualization and Virtual Reality (VR) techniques were least appreciated. Regarding the practical aspects of these tools, they found that the reliability of such a tool was classified as the most important aspect. They verified that code comprehension is considered the key for carrying out various maintenance and software life cycle tasks. Concerning code analysis aspects, only 3 out of 24 (desirable) aspects were identified as being supported by more than half of the tools. These aspects were: Visualization of function calls, of inheritance graphs, and of different levels of detail in separate windows. In the end, there is not a tool that fulfills all desires yet.

2.5.3 Automated Abstraction of Class Diagrams

A. Egyed [12] wrote an article that presents an approach for automated abstraction that allows designers to zoom out on class diagrams to investigate and reason about their bigger picture. Nevertheless, designers can easily become overwhelmed with details when dealing with (large) class diagrams. This approach that Alexander mentions is based on a large number of abstraction rules and, when used together, it can abstract complex class structures quickly. This article gives many examples about why class diagrams need to be abstracted and one of the reasons is that if you are only grouping the classes, it is still insufficient to achieve abstraction. A part of the abstraction rules are semantic rules in which you look at the semantic properties of classes and relationships which makes it possible to eliminate a helper class and derive a slightly more abstract class diagram. Another part of the abstraction rules consist of ambiguous model definitions. In total, the article provides 121 rules to abstract a class diagram. To date, they have validated their abstraction technique and its rules on numerous third-party applications and models with up to several hundred model elements. 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.

2.5.4 Reasoning on UML Class Diagrams

Berardi et al. [8] did a research study about reasoning on UML class diagrams. Their first contribution is to show that reasoning on UML class diagrams is EXPTIME-hard, even on restrictive assumptions; they prove this result by showing a polynomial reduction from reasoning in DLs. DLs are Description Logics, a family of logics that admit decidable reasoning procedures. Their second contribution consists in establishing EXPTIME-membership of reasoning on UML class diagram, provided that the use of arbitrary OCL (first-order) constraints is disallowed. Their third and final contribution has a more practical flavor and consists of polynomial encoding of UML class diagrams. They have shown in this paper that reasoning on UML class diagrams can be quite a complex task. They have proved that it is EXPTIME-complete, without considering arbitrary OCL constraints. This result suggests that it is highly desirable to provide automated reasoning support for detecting relevant properties of the diagrams. But there were some issues that remain to be addressed. One of them is that the reasoning tasks they had analyzed in this paper did not include reasoning on keys and identification constraints. But the experimentations they did, while certainly limited and not providing a definitive answer, indicate that current state-of-the-art DL-based systems are ready to serve as a core reasoning engine in advanced CASE tools.

3 Presence of Classes in Class Diagrams A Survey

A class diagram presents many classes, but do we need all of them? For a large sized and complex class a diagram there is a probability that there are classes that are not relevant. To discover this, we need to observe the classes by reviewing them based on the software design metrics of a class diagram. In this chapter, we present a survey that we have published online to enquire the information about which software design metrics are important. We have chosen 14 design metrics based on a tool called SDMetrics. We also present to the participants various flavors of class diagrams and question the participants about what classes should be excluded in these diagrams. Also, they were required to respond which class diagram flavor they preferred working with. In total, 25 complete responses were received with 76% having average or above skills with class diagrams. As the results, we found out that the metric that counts the number of public operations is the most important metric of them all. Also, we discovered that class names and coupling that is equal or less than 2 are influencing factors when it comes to excluding classes from a class diagram.

The outline of this chapter is the following: we first describe the survey methodology. Next, we show our results and give our findings based on these results. Then, we discuss our findings and give our conclusions based on this questionnaire's findings.

3.1 Survey Methodology

In this section we describe the design of the questionnaire in which we explain how the questionnaire was designed and why. We also give a description of our online survey experiment that explains how the experiment was conducted.

3.1.1 Questionnaire Design

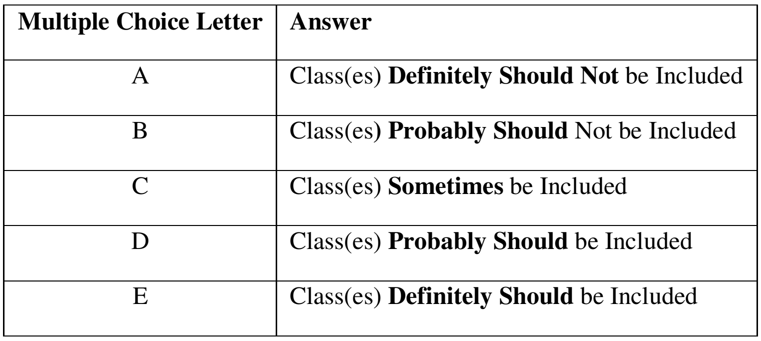
The questionnaire consisted out of 3 parts i.e. part A, B and C. There were a total of 24 questions in this questionnaire. In part A, we aimed to discover the respondent's personal characteristics and experience with class diagrams. In part B we aimed to discover what metrics the respondents find important when looking at a class diagram. As for part C, we aimed to discover what classes the respondents leave out when looking at a diagram and what diagram(s) the respondents prefer when looking at different diagram designs. This is an online questionnaire and is hosted by [2] and is available at [22].

3.1.1.1 Part A: Background of the Respondents

Part A consisted of 4 questions. In question 1, we asked about the current status of the respondents. Question 2 intended to collect the information about the respondent's location. This was an optional question, meaning that it is not mandatory for the respondents to answer this question. We asked how many years of experience the respondent has with class diagrams in question 3. In the last question of this part, we asked the respondent to rate their skills in creating, modifying and understanding class diagrams.

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

This part consisted of 14 questions. The first 13 questions asked about the aspects of a class diagram based on the software design metrics. In these 13 questions, the respondents were asked to answer the questions mainly about the indicators of class diagram inclusion based on design metrics. In each of the 13 questions, we briefly explained about the metrics that was used and 5 answers were offered for the respondents to choose. The choices of answers are shown in Table 3.



Multiple Choice Letter	**Answer**
A	Class(es) **Definitely Should Not** be Included
B	Class(es) **Probably Should** Not be Included
C	Class(es) **Sometimes** be Included
D	Class(es) **Probably Should** be Included
E	Class(es) **Definitely Should** be Included

Table 3: Answers Multiple Choice Questions

In the last question of part B (i.e. question 14), we tried to discover about the reason of the respondents in including and excluding a class in a class diagram. This question aimed to get the other information than software metrics about the reason of the respondents for including and excluding a class in a class diagram. This is an open-ended question and it is compulsory for the respondents to answer.

3.1.1.3 Part C: Practical Problems

Part C contained 6 questions. In this part, we tried to simulate some practical problem by providing several class diagrams. The class diagrams are derived from well-known domains i.e. Automated Teller Machine (ATM), Library System and Pacman Game. These well-known domain systems were selected to avoid bias about the domain knowledge of the respondents. The following class diagrams were involved in this survey:

1. **Automated Teller Machine (ATM) simulation system:** This class diagram is an ATM simulation example developed by the Department of Mathematics and Computer Science, Gordon College [9]. The ATM class diagram that was used in this survey does not contain any attributes or operations. With other words, only the classes with their class names and their relations with other classes are shown. In total, there are 22 classes in this class diagram.

2. **Library System:** The Library System is a system that enables a user to borrow a book from a library. This system is taken from [13]. The

system that we show in this questionnaire contains 24 classes and each class only shows their operations. The reverse engineered design was used for this questionnaire.

3. **Pacman Game:** Pacman's Perilious Predicament is a turn based implementation on the classic Pacman game. To accommodate its turn based nature, game play mechanics will be changed into more of a puzzle game. This project is found at [11]. In this questionnaire we used the diagram from the second phase or in this project called Milestone 2. We used two types of diagrams from this system namely the forward design and the reverse engineered design. The forward design consists of 17 classes while the reverse engineered design contains 15 classes. The reverse engineered design is created from the source and this source code is based on the forward design. With other words, the reverse engineered design is affected by the forward design.

We also tried to simulate the various flavors of class diagrams from the software industry by providing different Levels of Detail (LoD) of class diagrams and the sources of class diagrams. Different flavors of class diagrams allowed us to differentiate the indicators of class exclusion based on the class diagrams that were provided. The information about the class diagrams that we used in question 1, 2, 3 and 5 in part C is shown in Table 4.

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

Table 4: Description of the Class Diagrams Used in the Questions

Next to these 4 questions, we made another 2 questions in which we asked the respondent which class diagram he/she prefers. In the first question (question 4 in the survey) the respondent were required to choose between ATM system, Library system and the forward design of Pacman. The respondents were also required to provide the reason why they chose the answer.

In the second question (question 6 in the survey) the respondents were required to choose between the forward design and the reversed engineered design of Pacman. The respondents were also required to give the reason why they chose the answer. These questions (4 and 6) were provided with multiple choices of answers which the respondents were required to choose one of the answers. It was also mandatory to answer this question and the open-ended questions in which the respondents give the reason why they chose the answer. The multiple choice answers are shown in Table 5.

Multiple Choice Letter	**Answers question 4**	**Answers question 6**
A	I prefer class diagram A (ATM System)	I prefer class diagram C (Forward design Pacman)
B	I prefer class diagram B (Library system)	I prefer class diagram D (Reverse engineered design Pacman)
C	I prefer class diagram C (Forward design Pacman)	I prefer them Both
D	I prefer them all	I do not prefer them
E	I do not prefer them	It does not matter which one
F	It does not matter which one	*-not applicable-*

Table 5: Answers Part C Question 4 and 6

3.1.2 Experiment Description

The experiment was conducted online and was hosted by [2]. The questionnaire was published online from 15th of May until the 3rd of August. We first tried to invite students and researchers at the Leiden Institute of Advanced Computer Science (LIACS), Leiden, to our online questionnaire. Then, we promoted the questionnaire by using social media like Facebook, Twitter and Linked In. We also promoted this questionnaire to multiple online software developer forums. The questionnaire provided the facility to save the answers and the respondents could continue on a later time. The total respondents that entered this questionnaire were 98 (see Table 6). However, only 25 respondents completed this questionnaire. Most of the incomplete responses stopped after the questions in Part A.

3.2 Results and Findings

In this section we present our results and findings from this survey. This section is divided in three subsections: In the first subsection we show the results of part A of the questionnaire in which we accessed the background of the respondents. In the next subsection we present our results of part B of the questionnaire in which we asked the respondent's indication of class

No.	Responses	Amount
1	Complete Responses	25
2	Incomplete Responses	73
	Total Responses	98

Table 6: Total Reponses

diagram inclusion based on software design metrics. In the last subsection of this section we present the results of part C in which we asked the respondents some practical problems. The responses for this survey are available at [23].

3.2.1 Background of the Respondents (Part A)

In this subsection we present the results of part A of the questionnaire in which we accessed the respondent's background information. Part A of the questionnaire consisted of 4 questions and each question was analyzed. The results of each question of part A are the following.

3.2.1.1 Question A1: What is your status at the moment?

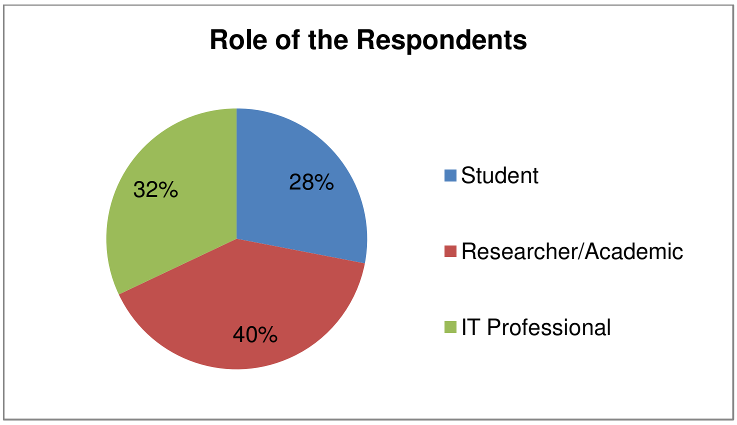


Figure 3: Role of the Respondents

In this question we asked the respondents what the respondent's status/role was at the moment. The respondent could choose out 4 answers which were the following: Student, Researcher/Academic, IT Professional, and Other.

In “Other”, the respondent could specify their status that differentiates from the previous three answers. The results are shown in Figure 3. 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 in this question. None of the respondents answered “Other” so Figure 3 shows the results of all the respondents. With these results we can conclude that the distribution of the respondent’s status is quite even.

3.2.1.2 Question A2: Indicate the location where you are currently working/studying

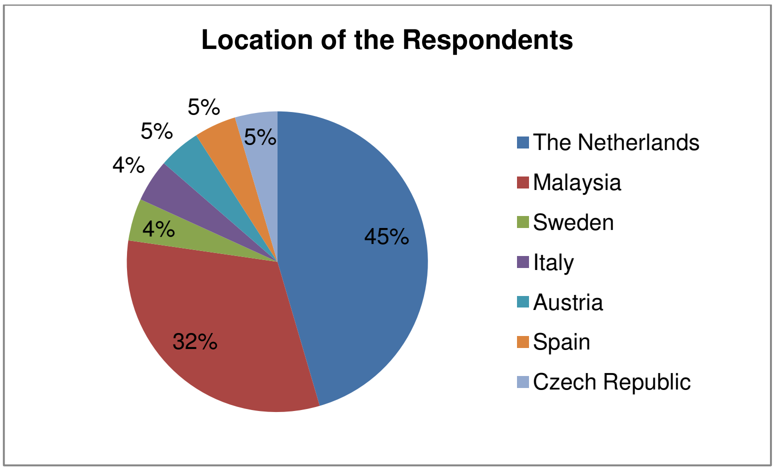
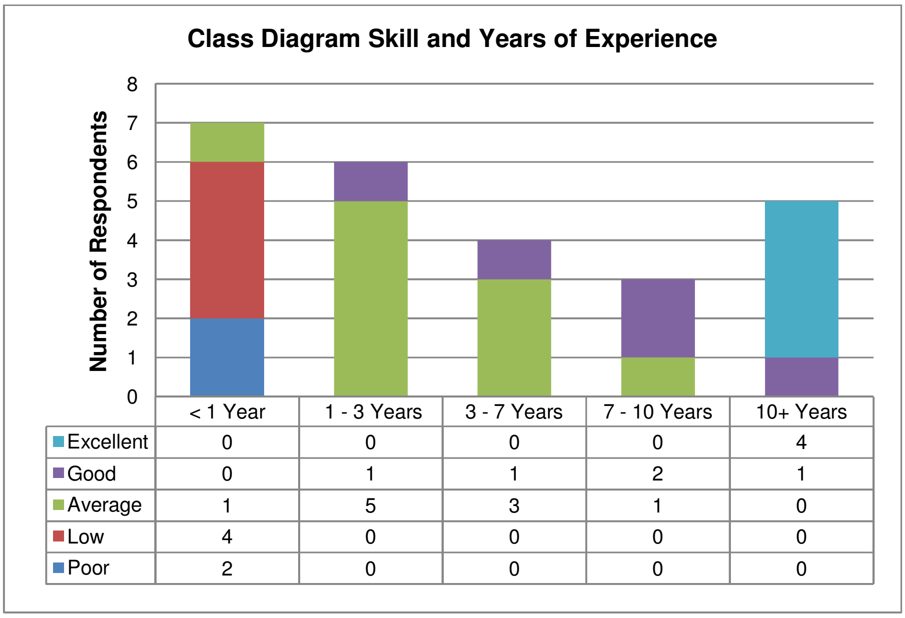


Figure 4: Location of the Respondents

In this optional question the respondent could state their location where they answered this questionnaire. This question was open-ended, meaning that the respondents were free to give any answer. Because of this, many respondents answered this question by stating their university for example. We looked in which country these universities were and added the country for it. The complete results of this question are shown in Figure 4. 45% of the respondents stated that they live in The Netherlands. 32% of the respondents are from Malaysia. This percentage is big because we asked some people from the Universiti Utara Malaysia, which is a university in Malaysia, to answer this online questionnaire. The rest of the respondents (4-5% each) accessed this questionnaire by seeing this in different forums which is posted by us.

3.2.1.3 Question A3: How many years of experience do you have in working with class diagrams? And Question A4: How do you rate your skills in creating, modifying and understanding a class diagram?



	< 1 Year	1 - 3 Years	3 - 7 Years	7 - 10 Years	10+ Years
🟦 Excellent	0	0	0	0	4
🟪 Good	0	1	1	2	1
🟩 Average	1	5	3	1	0
🟥 Low	4	0	0	0	0
🟦 Poor	2	0	0	0	0

Figure 5: Class Diagram Skill and Years of Experience

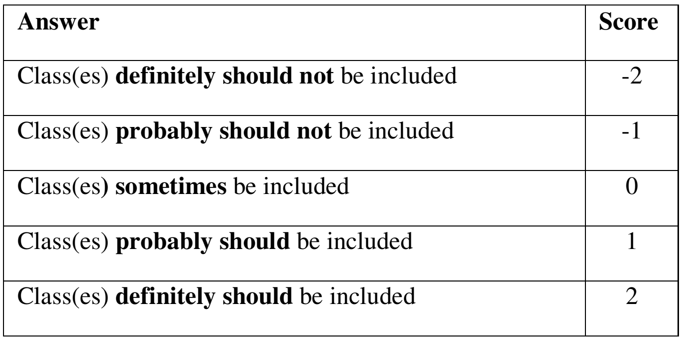
We combined the last two questions of part A to discover new findings and discuss about some matters which will be described later on.

Question A3 was about the experience the respondents have with class diagrams. The respondents could choose out of 5 answers which were the following: < 1 Year, 1 - 3 Years, 3 - 7 Years, 7 - 10 Years, and 10+ Years. 28% of the respondents stated that their experience with class diagram is less than 1 year. 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. As the results, it is quite evenly distributed. Even though many respondents mentioned that they have less than 1 year of experience, in the next question most of the respondents rated their selves that their skill is average or above, which is shown in Figure 5. This means that every respondent has knowledge about class diagrams.

In Question A4, we asked the respondent to rate his/her 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. 16% of the respondents and 8% of the respondents rated their skill Low and Poor, respectively. As the result, we can state that 76% of the respondents rated their skill of average or above. For the respondents that answered “< 1 Year” in the previous question, 6 out of 7 answered “Low” or “Poor” in this question. One of them answered this question with “Average”. This again means that (most of) the respondents have knowledge about class diagrams. The complete results of the combination of these two questions are shown in Figure 5.

3.2.2 Class Diagram Indicator for Class Inclusion (Part B)

In this subsection we present our result of part B. This part consisted of 14 questions. For question B1 to B13, the respondents were provided 5 answers to be chosen as their answer. The answers are shown in Table 3. We analyzed these 13 questions by using a score-system. The score system is shown in Table 7.



Answer	**Score**
Class(es) **definitely should not** be included	-2
Class(es) **probably should not** be included	-1
Class(es**) sometimes** be included	0
Class(es) **probably should** be included	1
Class(es) **definitely should** be included	2

Table 7: Score System Metrics - Question B1-B13

The reason of the scores is obvious: if a respondent does not want a class, it basically means that the metric that is in this class is not important and gets negative points. If a respondent answers with “Class(es) sometimes be included” then this respondent is neutral and the metric does not get any points. However, if a respondent answers that a class should be included, then the metric that is being asked in this question gains positive points.

Figure 6 shows the maximum and minimum of the scores that a metric can get. If a metric gets a negative score after going through all the answers then we can conclude that this metric is not important and that if a class contains a high frequency of this metric then this class must be excluded. If

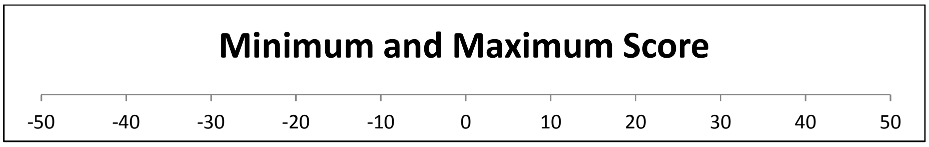


Figure 6: Minimum and Maximum Score

a metric has a positive score after going through all the answers then this metric is important and so is the class that contains this metric at a high frequency. However, if all metrics have a positive score, then this does not automatically mean that every metric is important. If this situation occurs, the metrics should be ranked based on the score and the metrics that are located amongst the highest position (ordered by highest to low) in the list are the important metrics. These important metrics should be presented in the class diagram and the low scoring metrics in the list are the metrics that should not be included in a class diagram.

The metrics are grouped in three categories which are: Size Category, Coupling Category, and Inheritance Category. Number of attributes (NumAttr), Number of Operations (NumOps), Number of Public Operations (NumPubOps), and Setters/Getters are grouped in the Size category. Outgoing and Incoming Dependencies (Dep_Out and Dep_In), Export Coupling Attributes and Operations (EC_Attr and EC_Par), and Import Coupling Attributes and Operations (IC_Attr and IC_Par) are grouped in the Coupling category. Number of Children (NOC), Depth in Tree (DIT), and Class in Leaf Depth (CLD) are grouped in the Inheritance category. The results of these three groups are presented in different subsections.

Meanwhile, in the last question of part B, we asked the respondent his/her reason of including or excluding a class in a class diagram. The analysis of this question will be described later.

3.2.2.1 Size Category (Question B1 - B4)

Figure 7 shows the results of the Size category. The Size category consists of four metrics namely NumAttr, NumOps, NumPubOps, and Setters/Getters. The highest score in this category has 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 state that the respondents found, if we only look at the size category, operations important in a class diagram, specifically operations that are public. This validates our results with our other questionnaire by looking at question C2B. In that question the results showed that most of the respondents did not like private and protected

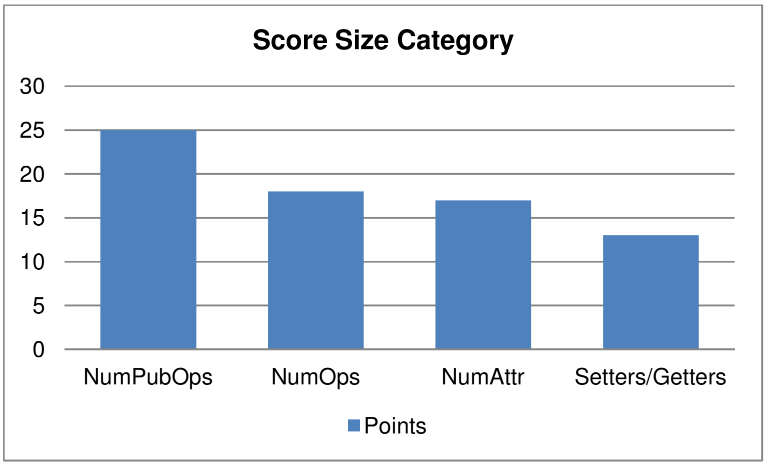


Figure 7: Score Size Category (Question B1-B4)

operations. In other words, they find public operations better and needs to be included in a class. As for setters/getters, it has the lowest points in this category. This indicates that the setters/getters are not an important element in a class diagram for the respondents. A reason for this could be that it is a common operation and also can be integrated in other operations that a system actually needs. NumAttr and NumOps also have a quite average amount of points. We can say that these metrics are normally needed in a class diagram but that public operations are more preferred.

3.2.2.2 Coupling Category (Question B5 - B10)

Figure 8 shows the results of the Coupling category. The coupling category consists of six metrics which are the following: Dep_Out, Dep_In, EC_Attr, EC_Par, IC_Attr, and IC_Par. As the results show, Dep_Out and Dep_In have 17 and 16 points, respectively. These metrics are one of the highest metrics in this category. In the survey in Chapter 4, many respondents find relationships in a class diagram important. Therefore, if a class contains many dependencies, whether they are outgoing or incoming, this class is important. This could be the reason that many respondents stated that such a class should be included. 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.

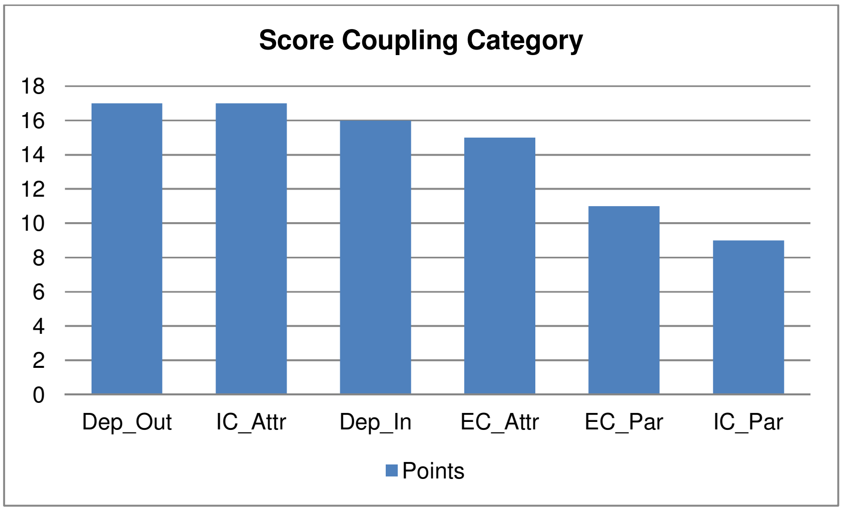


Figure 8: Score Coupling Category (Question B1-B4)

3.2.2.3 Inheritance Category (Question B11 - B13)

The Inheritance category consists of three metrics: NOC, DIT, and CLD. From the results that are shown in Figure 9, 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 conclude that the respondents only need a part of the inheritance tree. If a class has a high NOC then that means that this class is important because this class has many immediate children and is normally also high in the inheritance hierarchy. But if a class has a high DIT then this class is somewhere at the bottom of this hierarchy which means that there is a possibility that this class is not important. It is not a surprise that CLD has not many points because normally if a class has a high number of CLD then the class presents a very high level of abstraction that sometimes is used to group the classes under this class.

3.2.2.4 Class Inclusion/Exclusion (Question B14)

In this question we asked the respondents' reason of inclusion or exclusion of a class in a class diagram. To analyze this, we created several keywords that are related to the answers given to this question. This question can be divided into two answers: reason of including a class and reason of excluding a class. Most of the respondents give the reason on why they include a class in a class diagram but only 5 respondents have given the answer why they exclude a class. Thus, the keywords that we have made are specifically for the answers that said something about including a class. The answers that were about excluding a class will be summarized. There were 3 respondents

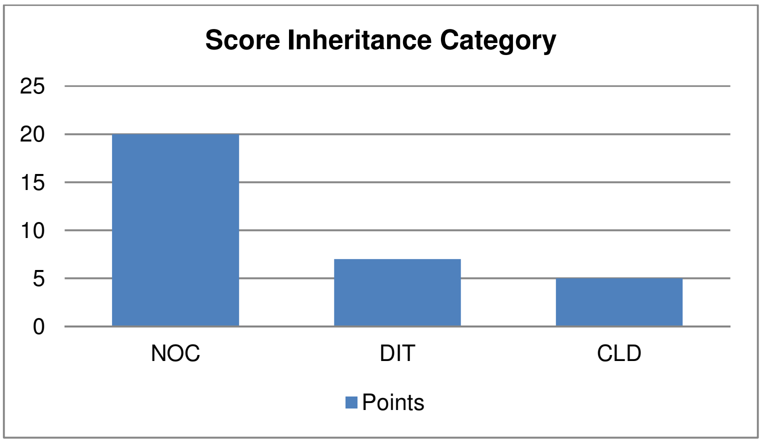


Figure 9: Score Inheritance Category (Question B1-B4)

that put an irrelevant answer in this question, “Not sure” for example. The keywords that we have used are the following:

- Size of Class/Diagram
- Complex Class
- Coupling
- Domain Related
- Understandability
- Frequent Class
- Based on Granularity
- Cohesion

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 respondents stated that they need a class when it is important but they do not say when such a class is important. This could be the weakness of this survey for not further questioning why the answer was given.

Figure 10 shows the results of the question based on the keywords. It shows 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%). Like we stated earlier, the keyword Important/Relevant class is a broad term but that is what the respondent answered with. So this answer is really obvious but we do not know when this class is important for them. Coupling on the other hand is the result that was expected. We stated in the survey in Chapter 4 that relationships are very important to understand a class diagram. Here, we found the same result.

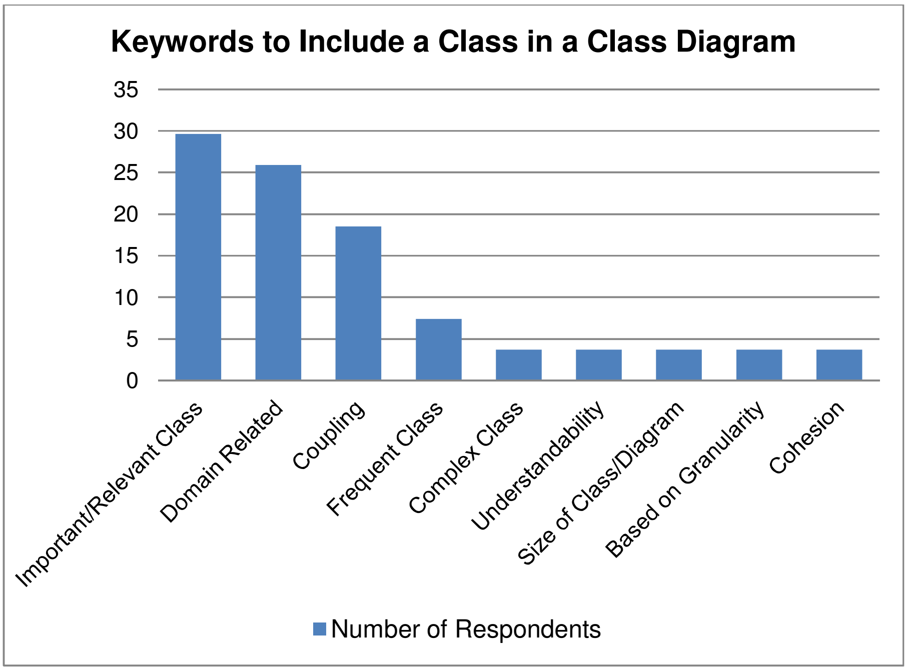


Figure 10: Keywords to Include a Class in a Class Diagram

18.5% of the respondents said that if a class has many relations then that class should be included. The last keyword is Domain Related. These are classes that are related to the concept or domain. Without these classes, it is hard for a software maintainer to understand a system. Thus, these classes must be included in a class diagram.

As we have mentioned earlier, only 5 respondents answered on the reason why they exclude class from a class diagram. One of these respondents stated that they would exclude a class when this class is too small and that it can be combined with another class. Another respondent stated that he/she excludes a class if this class does not contain any important attributes or operations. Once again, the respondent does not state when an attribute or an operation is important. One respondent stated that he does not need any children classes. With other words, he only needs the parent classes. Another respondent mentioned that he would keep the classes but would exclude the attributes and operations from these classes to get a high level abstraction. This answer is not really relevant to what we asked but it is interesting, because the respondent then only needs the class names and relations between the classes to understand a system. The last respondent stated that he/she excludes helper classes or technical-specific classes since they are not needed to understand a system.

3.2.3 Practical Problems

In this part, we tried to access the information about the classes that should not be included in a class diagram. The information is gathered by allowing the respondents to choose the classes that should not be included in a class diagram.

3.2.3.1 Question C1: Referring Figure 9, select the classes that you think should not be included in a class diagram

In this question, the ATM System class diagram was presented without attributes and operations (Figure 9 in the questionnaire). We expected to access the information about the influence of the coupling category and class names in a class diagram. The overall results of this question are shown in Figure 11.

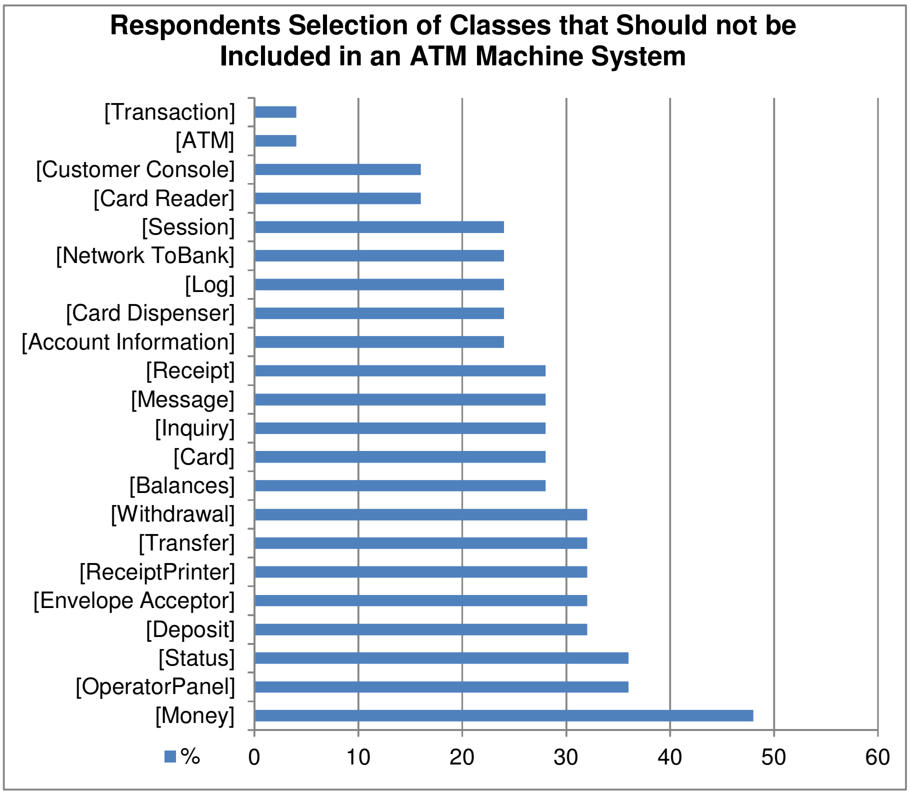


Figure 11: Respondents Selection of Classes that Should not be Included in an ATM Machine System

The results show that 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 <= 2. 32% of the respondents chose to exclude the classes Deposit, EnvelopeAcceptor, ReceiptPrinter, Transfer and Withdrawal. The coupling for those classes is equal to 2. This means 8 out of 24 classes in this class diagram were chosen to be excluded in a class diagram based on the amount of coupling.

The classes that were important in this class diagram are Transaction and ATM. The coupling in ATM is 9 and in Transaction it is 7. 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. In this question, we found that the meaningful class names seem not to be influenced much for the respondents. This is shown by 32% of the respondents that chose to exclude domain related classes i.e. Withdrawal, Transfer and Deposit. Those three classes are the functionality offered by the ATM Machine.

3.2.3.2 Question C2: Referring Figure 10, select the classes that you think should not be included in this class diagram

A reverse engineered class diagram from a Library System was used for this question (Figure 10 in this questionnaire). All elements in a class diagram were presented (HLoD) and we expected to discover the elements that influence in selecting the classes that should not be included. The results of the survey are shown in Figure 12.

From the results, it is obviously shown that most of the respondents chose not to include the classes that have no relationship. The top 7 classes that were chosen to be excluded in the class diagram were 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. The top three classes that were chosen to be excluded are AboutDialog, MessageBox and QuitDialog. From the class names, the respondents were able to predict what the functionality of the class is. AboutDialog, MessageBox and QuitDialog clearly mentioned the functionality of the classes that are used to display the information. Thus, these classes are not important because they are only used to display message. On the other hand, the 5 classes that not many respondents chose to exclude in a class diagram are classes that are related to the domain and have coupling more than 2. 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.

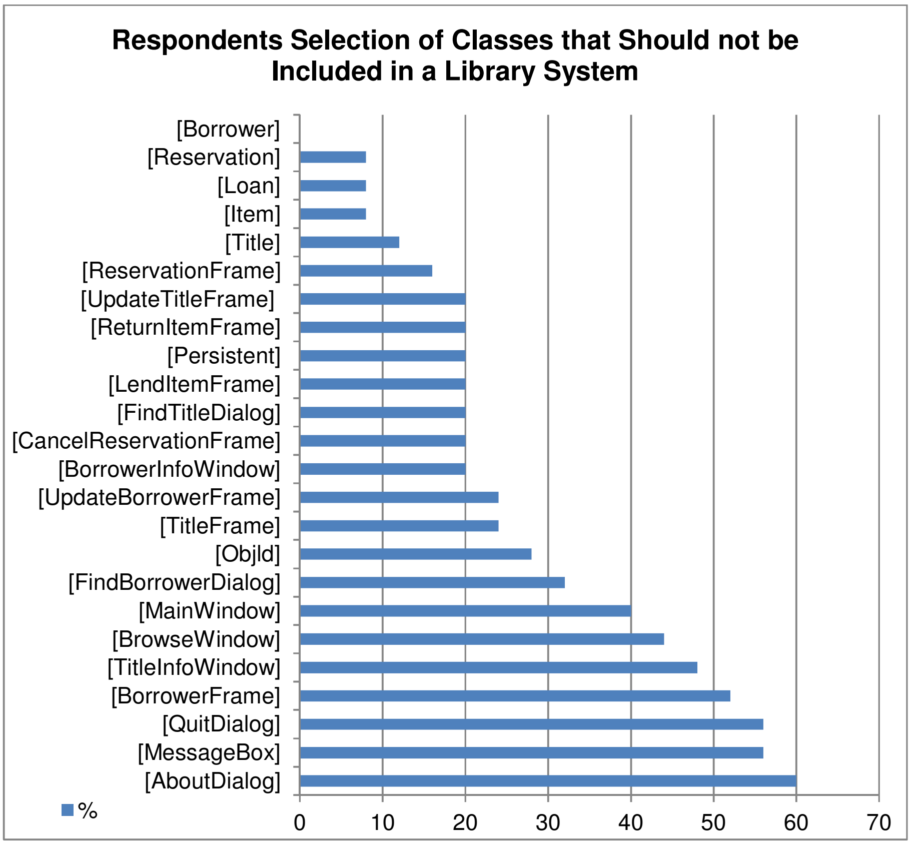


Figure 12: Respondents Selection of Classes that should not be Included in a Library System

From the analysis of the results, we found that meaningful class names and number of coupling influenced the selection criteria of important classes and not important classes for inclusion/exclusion in this class diagram.

3.2.3.3 Question C3: Referring Figure 11, select the classes that you think should not be included in this class diagram

In this question, the respondents were required to select the classes that are not important in a forward designed Pacman Game class diagram (Figure 11 in this questionnaire). Most of the classes in this diagram have relationships and meaningful class names. The complete result for this question is presented in Figure 13. The results indicate that 64% of the respondents chose class Direction to be excluded from a class diagram. This class is an Enumeration class and the coupling is equal to 0 which might be the reason why this class should not be included in a class diagram. 52% of the respondents selected to exclude the Iterator Class while 40% of the respondents chose not to include the Iterable Class. Both classes are only interface classes which might indicate that those classes are not important. PacShell only contains a main operation and might be common in programming. That may be the reason why this class has been chosen by 35% of the respondents to be left out from this class diagram.

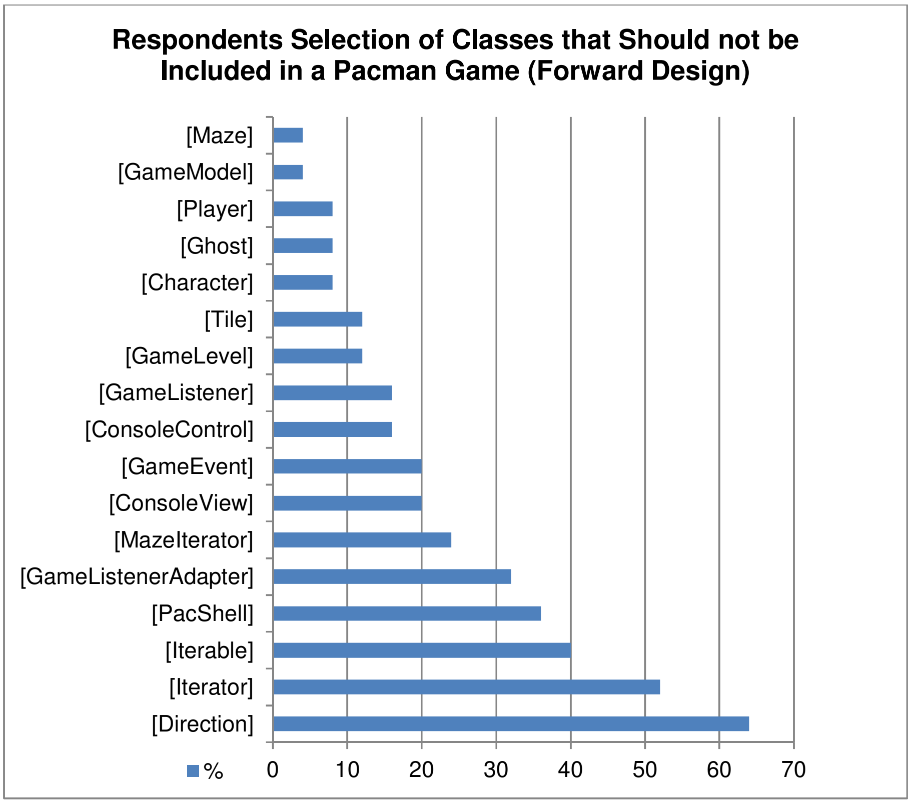
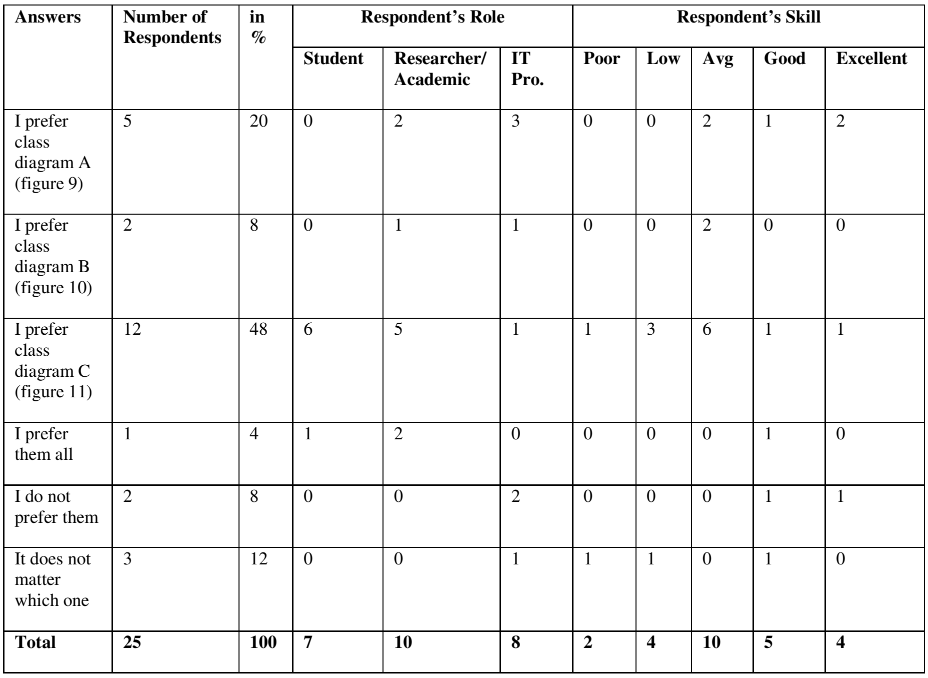


Figure 13: Respondents Selection of Classes that should not be Included in a Pacman Game (Forward Design)

From the results in this question, we found that the enumeration and interface types of classes are not important classes to be shown in a simplified class diagram.

3.2.3.4 Question C4: Referring to Figure 9, 10, and 11. Which class diagram do you prefer working with?

Referring to the class diagrams in C1, C2 and C3, the respondents have been asked which flavor of class diagram is preferred to be used. Class diagram in C1 presents Low Level of Detail (LLoD) in forward design while class diagram in C2 presents the reverse engineered design. Class diagram C3 presents High Level of Detail (HLoD) in a forward design. In this question, we tried to discover which class diagram is preferred by the respondents.



Answers	Number of Respondents	in %	Respondent's Role			Respondent's Skill				
			Student	Researcher/ Academic	IT Pro.	Poor	Low	Avg	Good	Excellent
I prefer class diagram A (figure 9)	5	20	0	2	3	0	0	2	1	2
I prefer class diagram B (figure 10)	2	8	0	1	1	0	0	2	0	0
I prefer class diagram C (figure 11)	12	48	6	5	1	1	3	6	1	1
I prefer them all	1	4	1	2	0	0	0	0	1	0
I do not prefer them	2	8	0	0	2	0	0	0	1	1
It does not matter which one	3	12	0	0	1	1	1	0	1	0
Total	**25**	**100**	**7**	**10**	**8**	**2**	**4**	**10**	**5**	**4**

Table 8: The Preferences between Class Diagram A, B and C

The results in Table 8 show that almost half of the respondents preferred working with class diagram C. This diagram is a HLoD forward design class diagram. 48% of the respondents preferred diagram C because they mentioned that the class diagram is clear, the necessary information are provided e.g. attributes and operations and most of the classes that are presented are important. This diagram was preferred most by students and researchers and one IT Professional. 20% of the respondents preferred to use class diagram A. 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. The respondents mentioned that they preferred this diagram because it is simple, less technical, domain oriented, systematic and has meaningful classes. 12% of the respondents mentioned that it did not matter which diagram they get while only 4% of the respondents preferred all the presented class diagrams. 8% of the respondents preferred class diagram B and another 8% did not prefer all the presented class diagrams. They did not prefer the class diagrams because they mentioned that there is “no story” in the class diagrams and the class diagrams only show the solution, not the foundation of the domain.

3.2.3.5 Question C5: Referring Figure 12, select the classes that you think should not be included in this class diagram

This class diagram was derived from the domain of a Pacman Game. It is slightly different with the class diagram presented in question C3 because this class diagram was constructed by using a reverse engineering technique. The Pacman Game implementation is closely following the forward design and that is the reason why there is a small difference between the forward engineered class diagrams and the reverse engineered class diagram. 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 that is close or almost similar with the forward design class. We also tried to discover which class diagram was preferred which was asked in a later question.

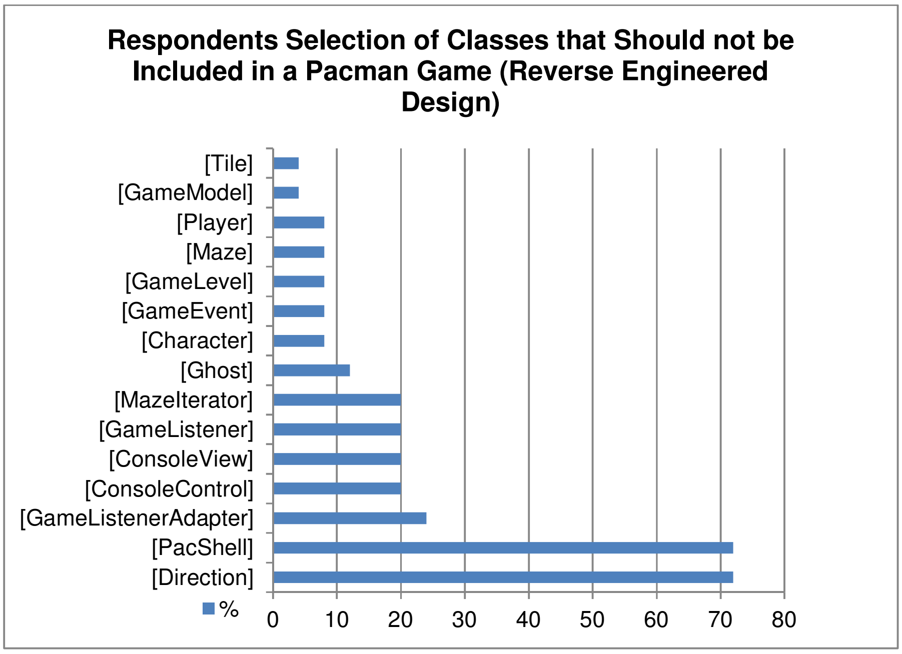
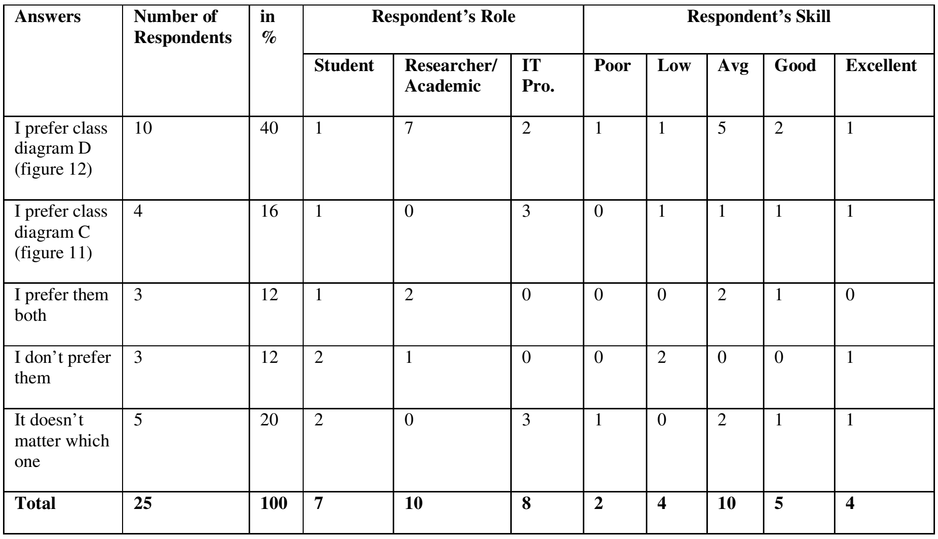


Figure 14: Respondents Selection of Classes that should not be Included in a Pacman Game (Reverse Engineered Design)

The complete result of this question is shown in Figure 14. The result shows that the class Direction and PacShell were selected by 72% of the respondents to be left out from the class diagram. The reasons could be that those classes have no relationship to other classes, it is an enumeration class (Direction) and it is a common programming class (PacShell). 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. For instance, the interface class Iterator is presented in the Maze class. This result shows that coupling influenced the selection of a class to be excluded in a class diagram.

3.2.3.6 Question C6: Referring Figure 11 and 12, which class diagram do you prefer working with and why?

From question C4, the result shows that most of the respondents prefer to use the forward engineered class diagram. This 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 is different with the reverse engineered class diagram in question 2 because this class diagram (figure 12) was derived from a system that was implemented (or coded) closely with the forward design.



Answers	Number of Respondents	in %	Respondent's Role			Respondent's Skill				
			Student	Researcher/ Academic	IT Pro.	Poor	Low	Avg	Good	Excellent
I prefer class diagram D (figure 12)	10	40	1	7	2	1	1	5	2	1
I prefer class diagram C (figure 11)	4	16	1	0	3	0	1	1	1	1
I prefer them both	3	12	1	2	0	0	0	2	1	0
I don't prefer them	3	12	2	1	0	0	2	0	0	1
It doesn't matter which one	5	20	2	0	3	1	0	2	1	1
Total	**25**	**100**	**7**	**10**	**8**	**2**	**4**	**10**	**5**	**4**

Table 9: The Preference between Class Diagram C and D

The results in Table 9 show that most of the respondents (mainly researcher) preferred to use the reverse engineered class diagram (Class Diagram D). 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. The reason mentioned by these respondents was that both class diagrams are equally good and similar. On the other hand, 16% of the respondents preferred class diagram C. The respondents mentioned that class diagram D has a complete view of the attributes and operations and all classes are successfully linked. There is no pattern of selection presented in this result in terms of the respondents' role and skill.

If we compare the results of this question and the results of question C4, we found that the reverse engineered class diagram is chosen if the source code was closely implemented based on the forward design. These kinds of source codes are capable to construct a very helpful class diagram that is mostly comparable with the forward design class diagram and are sometimes even better. This means that reverse engineered class diagrams that are mostly similar with the forward design are preferred for the software engineer to understand a system design.

3.3 Discussion

In this section we discuss the results and findings presented in the previous section. This section is divided into 5 parts which are the following: Respondents' Background, Software Design Metrics, Class Names and Coupling, Class Diagrams Preferences, and Threat of Validity.

3.3.1 Respondents' Background

In Part A, we have accessed the respondents' background information. The information is about the respondents' status and also skills and experience in working with class diagram. As the result, the respondent's status in this questionnaire was quite evenly distributed. We also asked about the location of the respondents and the result 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. The respondents that have less than one year experience also answered that they have low or poor skill aside from one respondent that rated average. With these results we can confirm that the respondents are truthful to their experience and their skills regarding to class diagrams.

This also confirms that every respondent had the minimum knowledge to answer this questionnaire.

3.3.2 Software Design Metrics

In Part B, the respondents were asked about the indicators of a class to be included in a class diagram based on the selected software design metrics. The analysis was done based on the Size, Coupling and Inheritance categories. In the Size category, we found that the higher the number of public operation is the higher the possibility of a class is to be included in a class diagram. 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 more important than operations in general which also contains private and protected operations. The Setters/Getters metric had the lowest points (13 points). Setters and getters are operations that are used to access value (getters) and update or modify value (setters) of attributes in a class. These kinds of operations can be merged into operations that are needed for the system to function. Thus, if a class has many getters and setters then this class should not be included in a class diagram.

In the Coupling category, we have discovered that classes that have many incoming and outgoing dependencies are important since the points that they have are 17 and 16, respectively. These points are not high if you compare it with the other categories but in this category these two metrics are one of the highest, next to IC_Attr with also 17 points. We thus discovered that dependencies are important. Dependencies are also relationships and if we look at the results of the next part of the questionnaire we see that coupling (with other words: relationships) is an influencing factor if we want to include or exclude a class. We have mentioned earlier that attributes are a common element in a class. Here, we found that IC_Attr and EC_Attr have a high amount of points (17 and 15, respectively). They have more points than EC_Par (11 points) and IC_Par (9 points). The reason might be that the class that is declared as an attribute is more important because the class could be used for every operation in the class. Meanwhile, if the class is only declared as parameter in an operation, the object of the class can only be used by the operation internally.

In the Inheritance category, we discovered that for a class that has a high number of children (NOC), the class should be included in a class diagram. This parent class is helpful to show the abstraction of a group of classes. On the other hand, DIT and CLD show the lowest scoring among the software design metrics. 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 then this means that this class is very abstract, meaning that this class alone will not be enough to understand the whole hierarchy. So it is basically a class that presents an abstraction of classes.

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. This means that all software metrics that were used in this study is useful. A negative score means that the class is not useful and should not be included in a class diagram. As we mentioned before, the main purpose of this study is to get the important metrics that influence the class inclusion in a class diagram. Based on the result, we can get this information by ranking the score of these metrics. The overall ranking of the score is shown in Table 10. This result could be applied for a software designer to simplify a class diagram during the documentation phase. This result is a little bit contradicting with the result in Part C. In Part C, a lot of metrics that are related to relationship have a higher score than the metrics from the Size category. Part C shows that relationship is an important element in a class diagram.

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

Table 10: Overall Score for Software Design Metrics

3.3.3 Class Names and Coupling

Based on the results in Part C, we discovered that a very influencing factor when we are trying to exclude classes from a class diagram is coupling. Based on the results, we have seen that most of the respondents exclude classes that have no coupling at all, meaning that these classes does not have any relationships. Many respondents also exclude classes that have coupling <= 2. Another influencing factor is the class name. Many respondents excluded 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 actually excluded domain related classes, classes that are needed for the functionality of the ATM system.

Aside from these two big influencing factors, many respondents excluded the type of classes like enumeration and interface. Either of these classes did not contain any information in it or the coupling was very low. Another reason of why the interface classes are excluded could be that these classes are GUI related.

3.3.4 Class Diagram Preferences

In question C4, the respondents were required to choose between forward design (LLoD), reverse engineered design (HLoD) and forward design (HLoD). Based on the results, 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, the respondents were required to choose between the forward design (HLoD) and reverse engineered design (HLoD) of the same system. In this question, the result was different from the result in question C4. Most of the respondents had chosen the reverse engineered design (HLoD). The reason might be that the reverse engineered design that was provided for this question was almost similar with 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.

From our observation, the reverse engineered class diagram of the Library system was not preferred because the structures of the classes were not well-presented. This might be because the implementation was not conforming to the design or there was no design in the system before implementation.

3.3.5 Threat of 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 also not well distributed in this survey because most of the respondents came from The Netherlands and Malaysia. Most of the questions in this study require the respondent to choose the best answers. We needed to do predictions on why the respondents chose these answers and this prediction may not be accurate. This questionnaire should contain question in which it asks why the respondent chose the answer to get the reason.

3.4 Conclusion

In this survey we have discovered the most important elements in a class design that should be included in a class diagram. We also discovered what flavor of class diagrams is preferred to work with. We discovered these findings by doing an online questionnaire. There were 25 respondents that completed this questionnaire.

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. Classes that have number of coupling less or equal to 2 are most likely to be excluded from the class diagram. Our most significant discovery of this survey is the preference of class diagrams the respondents had. The reverse engineered design is being preferred over the forward design. However, the source code must implement the forward design so that the reverse engineered design is mostly similar to the forward design.

With these results we can now highlight the reverse engineered class diagrams if they are good for understanding a system or not. We can also highlight which classes should be included or excluded based on our results and analysis by looking at the metrics and behavior 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 when selecting a class to be included or excluded in a class diagram and we discovered what type of class diagrams these respondents prefer which are some important elements that could be used for simplifying UML class diagrams.

4 Class Diagram Simplification: What is in the developer's mind?

Class diagrams are diagrams that should support the software developer in understanding a system. However, sometimes the class diagram is too complex to easily understand a system in a short period of time. Is there a way to simplify this diagram? This survey is to enquire the information about what type of information they would include or exclude in order to simplify a class diagram. This survey involved 32 software developers with 75 percent of the participants having more than 5 years of experience in class diagrams. We discovered various elements that are important in a class diagram such as relationship, class names and properties. We also discovered elements that are not important in a class diagram such as GUI related information, private and protected operations, and constructors without parameters.

The outline of this chapter is the following: we first describe the survey methodology. Then, we show our results and give our findings based on these results. We end this chapter by discussing our findings and presenting our conclusions based on our analysis of this questionnaire.

4.1 Survey Methodology

In this section, we describe i) The questionnaire design that explains how the questionnaire was designed and why; and ii) The experiment description which explains how the experiment was conducted.

4.1.1 Questionnaire Design

The questionnaire was organized into three parts i.e. Part A, B and C. In total, there were 15 questions. In Part A, we aimed to discover the information about the respondent's personal characteristics, knowledge and experience with UML class diagrams. Meanwhile in Part B and C, we aimed to discover the information about how the respondents indicate classes that should be included in a class diagram. For this survey, we organized the questionnaire by dividing this questionnaire into 2 different sets of questions. Both sets of questions had the same questions for Part A and C. However, we differentiated the questions in Part B. The questionnaire can be found at [20].

4.1.1.1 Part A: Personal Questions

Part A consisted of six questions. Questions 1 to 4 in this questionnaire were intended to access the information about the status of the respondents, years of working with class diagrams, where they learned UML, and how the respondents rate their skills in class diagrams. In questions 5 and 6, we wanted to compare the respondents' preferences for UML models or source code for understanding a system.

4.1.1.2 Part B: Practical Problem

This part contained 3 questions and each question consisted of a class diagram. In this part, the respondents were required to mark information that that can be left out in provided class diagram without affecting their understanding of the system. They were also allowed to write any comments or suggestions according to what information they find unnecessary in a class diagram. The following class diagrams were involved in this survey:

1. **Automated Teller Machine (ATM) simulation system:** This fully functional system has a class design and complete implementation source code. The class design was made by using forward design. The case study is an ATM simulation example developed by the Department of Mathematics and Computer Science, Gordon College [9]. This simple simulation system is used to show the overall process of UML usage in analysis, design and implementation phase. The complete software documents based on UML that were provided consists of 22 design classes. We reverse engineered the design of this system for this study.

2. **Pacman Game:** Pacman's Perilous Predicament is a turn based implementation on the classic Pacman arcade game. To accommodate its turn based nature, gameplay mechanics will be changed into more of a puzzle game [11]. In this survey, we used the diagram from the 2nd phase or in this project called Milestone 2. The amount of classes in the source code in this system is 17 while only 15 classes are stated in the class diagram design. Both forward and reverse engineered designs were used in this survey.

3. **Library System:** Library System is a system that enables a user to borrow a book from a library. This system is taken from [13]. This complete system consists of 24 classes in the source code. The reverse engineered design was used for this survey.

As mentioned, this survey consisted of two different set of questions. This part differentiates the set of questions by providing different types of class diagrams. The information about the sets of class diagrams are shown in Table 11.

Every set of the questionnaire had both MLoD and HLoD. In set A, ATM system in MLoD and Library System in HLoD were used and in set B, ATM system in HLoD and Library System in MLoD were used. Different Level of

No	Class Diagram	Set A	Set B
1	ATM System	Medium Level of Detail (MLoD)	High Level of Detail (HLoD)
2	Pacman Game	Forward Engineered Design	Reverse Engineered Design
3	Library System	High Level of Detail (HLoD)	Medium Level of Detail (MLoD)

Table 11: Information on Set A and Set B

No	Class Diagram Elements	Medium Level of Detail (MLoD)	High Level of Detail (HLoD)
1	Classes	YES	YES
2	Attributes	YES	YES
3	Type in Attributes	NO	YES
4	Operations	YES	YES
5	Operations Return Type	YES	YES
6	Parameters in operation	NO	YES
7	Relationships	YES	YES

Table 12: Level of Detail Description

Detail (LoD) were used to simulate different types of details that normally exist in a class diagram. We also used different sources of class diagrams by setting forward design and reverse engineered class diagrams to simulate the different flavors of class diagrams that exist in the software industry. Table 12 explains about the Level of Detail.

4.1.1.3 Part C: Class Diagram Indicators for Class Inclusion

This part consisted of six open-ended questions. The aim of these questions was to discover what the developers think about the information that is needed in a class diagram and the information that should be left out. Table 13 describes the questions in part C.

4.1.2 Experiment Description

The experiment was conducted on 6th of June 2012 at Leiden Institute of Advanced Computer Science (LIACS), Leiden. The participants of this survey were software developers from all over the Netherlands. In total, there

No.	Question	Description
1.	**Question C1**: In software documentation, particularly in class diagrams, what type of information do you look for to understand a software system? (for example: relationships, operations, attributes, etc)	To learn what type of information is important to understand the software system.
2.	**Question C2**: In a class diagram, what type of information do you think can be left out without affecting your understanding of a system? a. Classes (for example: Helper class, Interface class, Library class, ….) b. Operations (for example: private, protected, public, constructor, ….) c. Relationships (for example: labels, multiplicities, self-relations) d. Other(s):	To find out what type of information can be left out from a class diagram.
3.	**Question C3**: Do you think that a class diagram should show full hierarchy of inheritance? If not, which parts could be left out? (for example : parent, child, intermediate parent/child, leaf, …)	To find out what type of information in the inheritance relationship is important.
4.	**Question C4**: What criteria do you think indicate that a class (in a class diagram) is important for understanding a system?	To discover how the developers recognize the criteria of a class that is important in a class diagram.
5.	**Question C5**: If you try to understand a class diagram, which relationships do you look at first? (Example: dependencies, inheritance, associations, etc)	To determine which relationship that can be considered important in a class diagram.
6.	**Question C6**: If there is a tool for simplifying class diagrams (e.g. obtained from reverse engineering), what features\functions would you expect from such a tool?	To find out what kind of features or functions are needed for a class diagram abstraction tool.

Table 13: Detailed Explanation Part C

were 32 respondents and all of them were a member of a survey community called Devnology [17]. The participants had to answer every question and were free to ask any questions during the questionnaire session. The time given to answer the questionnaire was 60 minutes.

4.2 Results and Findings

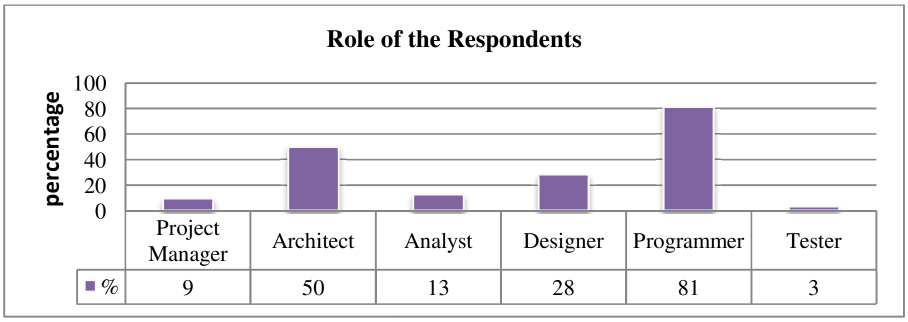
In this section you will find our analysis and results of the answers given by our respondents. We have split this section up in three parts. In the first part we give our findings of part A of the questionnaire in which we asked the respondents some personal questions. In the second part we give our findings of part B of the questionnaire which consisted of the practical problems. In the last part we give our findings of part C of the questionnaire. A full explanation of these parts is given in section 2: Survey Methodology and the responses for this survey are available at [21].

4.2.1 Part A: Personal Questions

This part consists of six questions related to personal characteristics, knowledge and experience. We will give our findings on each question in this part as well as the other parts. In part "Others" we present several combinations of the results.

4.2.1.1 Question A1: What is your role at the moment?

In this question the respondent should state his/her role in software development. The choices of answers that have been given to the respondents are Project Manager, Architect, Designer, Programmer, and Tester. The respondents were allowed to select more than one answer.



	Project Manager	Architect	Analyst	Designer	Programmer	Tester
■ %	9	50	13	28	81	3

Figure 15: Role of the Respondents

As for the results, 81% of the respondents are programmers and half of the respondents are software architects. As shown in Figure 15, 28% of the respondents are software designers. Figure 15 also highlights that the majority of the respondents are involved in the Design and Implementation phase in software development. 14 out of 26 programmers (54%) are also software architects or software designers. This means that half of the programmers are involved in designing the software. All project managers that were involved in this study are also programmers. This indicates that all the respondents that participated in this study are directly involved in software development.

4.2.1.2 Question A2: How many year(s) of experience do you have in working with class diagrams?

This question is about the experience of the developer in working with class diagrams. Out of 32 respondents only 28 (88%) of the respondents answered this question. Figure 16 shows the complete results of this question. From these results we found that 50% of the respondents are experienced with class diagrams for more than 10 years. This is expected because most of the participants of this survey indicated that they know UML when we asked them before the questionnaire was handed over. The results also show that 75% of the respondents have experience with class diagrams for more than 5 years. There are only about 11% (3 respondents) having less than 1 year experience in class diagrams. Even though they have less experience in class diagrams, they have knowledge about UML based on the results in Question A3.

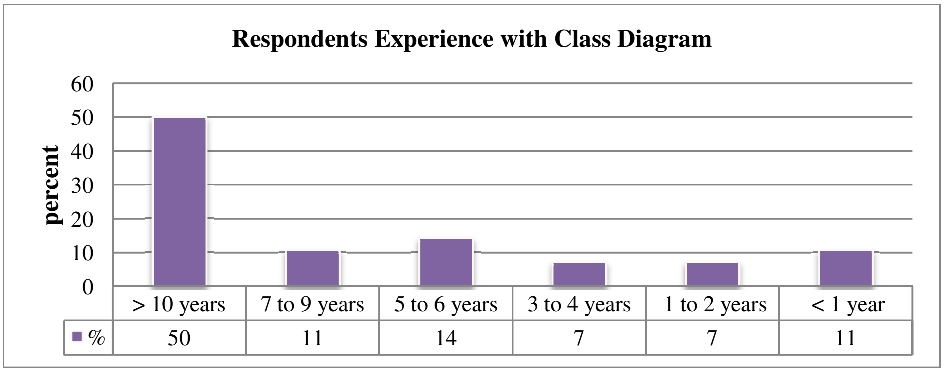


Figure 16: Respondents Experience with Class Diagrams

4.2.1.3 Question A3: Where did you learn about UML?

This question asks about where the respondent learned about UML. We expected to discover where the respondent learned about UML and also whether all the respondents know about UML or not. In this question, the respondents were allowed to choose more than one answer. The choices were the following: Did not learn UML, From Colleagues/Industrial Practice, Professional Training, Learn by Myself, and HBO/University. The results show that 47% of the respondents had learned about UML in HBO or University and 25% have taken professional training to learn UML. This indicates that 72% of the respondents had formal training of UML. Meanwhile, 38% of the respondents learned UML by themselves and 19% learned from their colleague(s) or from industrial practice. There were no participants that answered ‘No’. This shows that all participants of this survey have knowledge of UML. Figure 17 shows the complete results of this question.

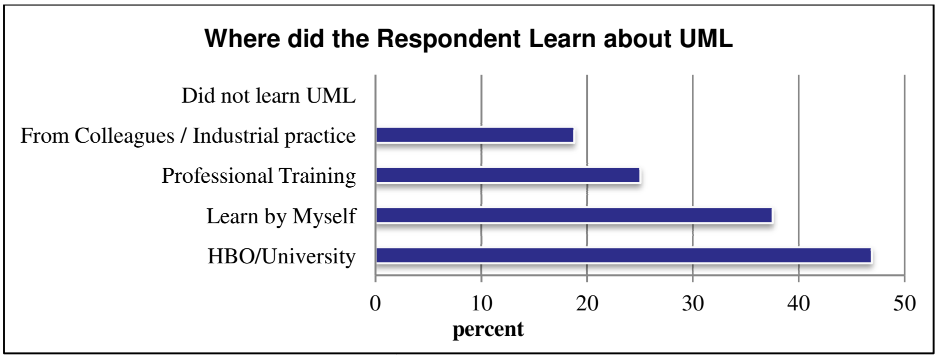
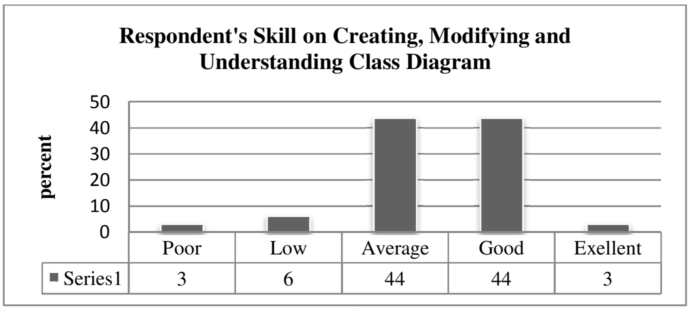


Figure 17: Where did the Respondent Learn about UML

4.2.1.4 Question A4: How do you rate your own skill in creating, modifying and understanding a class diagram?

This question was aimed to gain knowledge about the skills of the respondents in creating, modifying, and understanding class diagrams. Based on Figure 18, most of the respondents (88%) have average or good skills on creating, modifying, and understanding class diagrams and only 3% have excellent skills related to class diagrams. This indicates that over 90% of the respondents have average skills or above related to class diagrams. Meanwhile, 2 respondents (6%) have low skills and only 1 respondent (3%) has poor skills related to class diagrams. The 2 respondents that have low skills are software architects (with no other role) and the only one respondent that has poor skills is a programmer (with no other role).



	Poor	Low	Average	Good	Exellent
■ Series1	3	6	44	44	3

Figure 18: Respondent's skill on Class Diagram

4.2.1.5 Question A5: Indicate whether you (dis)like to look at source code for understanding a system? + Question A6: Indicate whether you (dis)like to look at UML models for understanding a system?

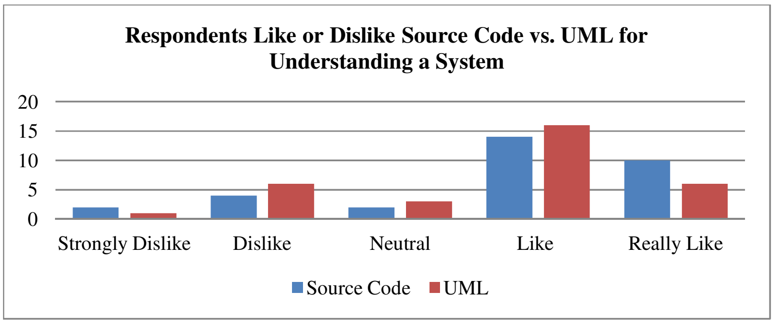


Figure 19: Respondents Like or Dislike Source Code vs UML

Question A5 and A6 was aimed to discover the respondent's opinion about the usage of UML and Source code as an artefact to understand a system. From the nature of the respondents whereas most of the respondents of this survey are programmers, we expected that the respondents would choose source code over UML. To present this result, we combined these two questions for a comparison between the respondent's like or dislike for UML and the respondent's like or dislike for source code. The results shown in Figure 19 indicates that in general there is no significant difference between Like or Dislike of Source code versus UML design. We may say with this overall result that even experienced programmers found that UML is helpful for understanding a system. We further investigated this result by separating this according to the role of the respondents, specifically programmer, software architect, and software designer.

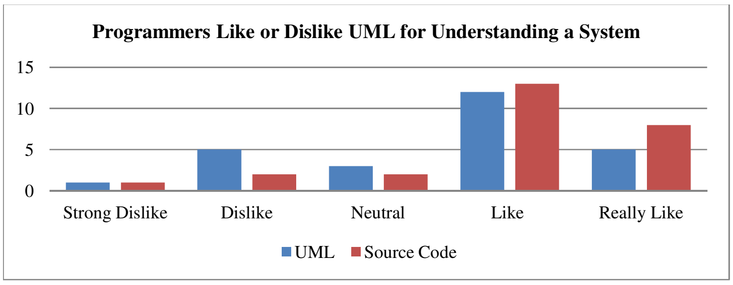


Figure 20: Programmers Like or Dislike Source Code vs UML

Figure 20 shows the results of question A5 and A6 for respondents with the role of a programmer. The results show that the programmers are a bit more positive about source code than UML but the difference is not significant. These results seem almost the same with the overall results shown in Figure 19. These results were expected because an experienced programmer could understand source code in a short time limit.

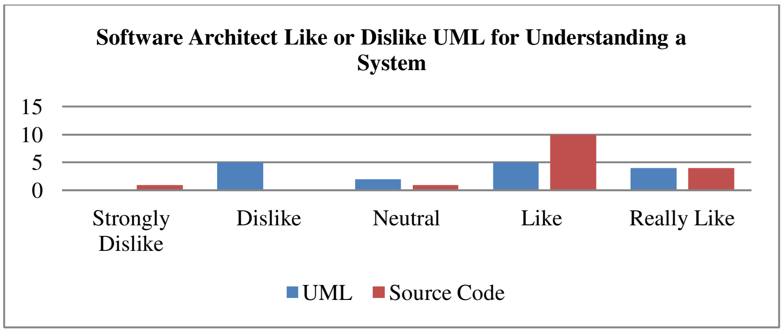


Figure 21: Software Architects Like or Dislike Source Code vs UML

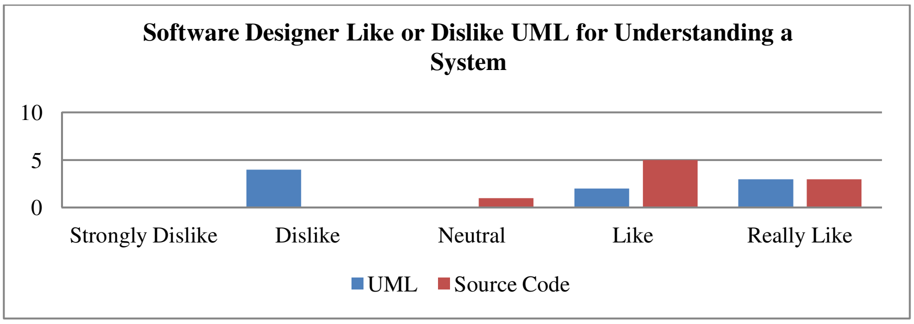
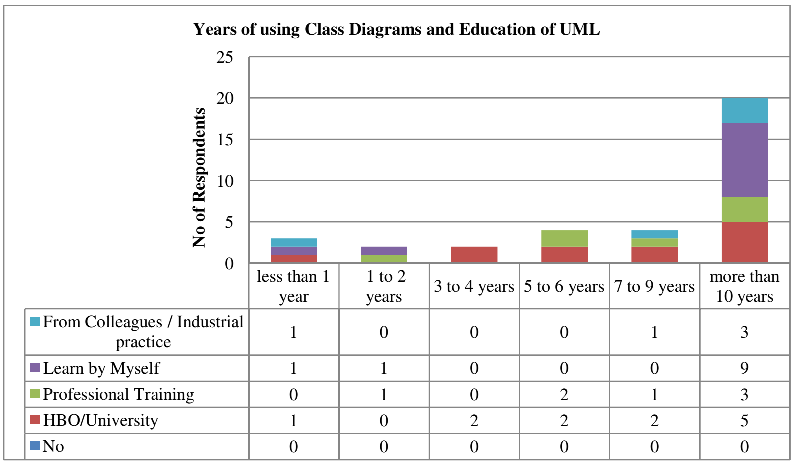


Figure 22: Software Designers Like or Dislike Source Code vs UML

It was quite a surprise to see that a lot of software architects like using source code more than UML to understand a system (Figure 21). The same goes for the software designers, they like using source code more than UML to understand a system (Figure 22). However, these results may not be purely accurate because as we can see in question A1 (role of respondents), most of the designers and architects in this survey are involved in development or are also a programmer.

4.2.1.6 Others:

Combination of Question A2 & A3



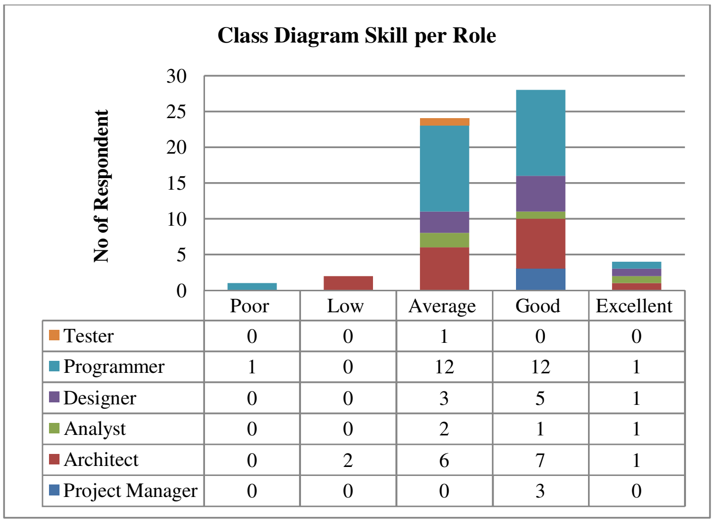
	less than 1 year	1 to 2 years	3 to 4 years	5 to 6 years	7 to 9 years	more than 10 years
🟦 From Colleagues / Industrial practice	1	0	0	0	1	3
🟪 Learn by Myself	1	1	0	0	0	9
🟩 Professional Training	0	1	0	2	1	3
🟥 HBO/University	1	0	2	2	2	5
🟦 No	0	0	0	0	0	0

Figure 23: Years of using Class Diagrams and Education of UML

Figure 23 combines the answers given on question A2 with the answers given on question A3. This figure shows that 45% of the respondents with 10 years of experience and above learned UML by themselves. Also, most of the respondents that answered "Learned by myself" came from this group. The respondents that answered HBO/University are more spread out over the years of experience and this option has been answered the most if we look at question A3. Figure 23 also proves that all respondents in this survey have minimum knowledge of UML even though there are respondents that have answered that they have experience in UML for less than one year.

Combination of Question A1 & A4

The combination of results in question A1 and A4 is shown in Figure 24. Overall, if we consider that the minimum skill on class diagrams is Average; over 90% of the respondents have the skill of creating, modifying, and understanding class diagrams. The only respondent that has poor skills in class diagrams is a programmer (no other role) and both respondents that have low skills are software architects (with no other role). This result surprisingly shows that there were software architects that rated themselves poor in creating, modifying, and understanding class diagrams. However, based on our informal interview with these respondents, a software architect mentioned that they only use boxes and lines for their architectural work. This may be the reason why there are software architects that have a poor skill in class diagrams.



	Poor	Low	Average	Good	Excellent
🟧 Tester	0	0	1	0	0
🟦 Programmer	1	0	12	12	1
🟪 Designer	0	0	3	5	1
🟩 Analyst	0	0	2	1	1
🟥 Architect	0	2	6	7	1
🟦 Project Manager	0	0	0	3	0

Figure 24: Class Diagram Skill per Role

4.2.2 Part B: Practical Problems

In Part B, the respondents have been provided with three class diagrams from different systems and domains. For all class diagrams they were asked to mark or give any suggestions about what can be excluded in the class diagram without affecting the understanding of the system. The results of this part were analyzed by combining the answers based on the following categories: Attribute, Operation, Class, Relationship, Inheritance, Package, and Others.

4.2.2.1 Category 1: Attribute

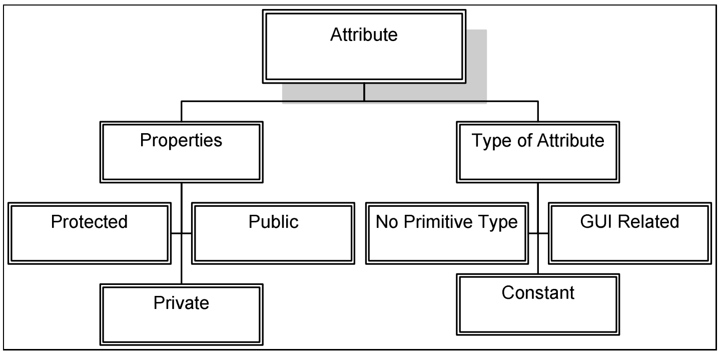


Figure 25: Attribute Category

In the Attribute category, we divided this category into two subcategories: Properties and Type of Attribute. We divided the Properties subcategory in three elements: Protected, Public and Private. This basically means that if a respondent marked the private variables in a class diagram or mentioned about excluding the private attribute, we considered that the respondent chose not to include the Private attribute element in a class diagram. The same goes for the other elements in this subcategory. We also divided the Type of Attribute subcategory into three elements: No primitive type, GUI related, and Constant. No primitive type is an attribute that does not have any primitive type. GUI related attributes are attributes that are related to Graphical User Interface (GUI) libraries that are provided by the development tools such as Textbox, Label and Button. Constant variables are variables that cannot be changed. Figure 25 illustrates subcategories and elements in the Attribute category.

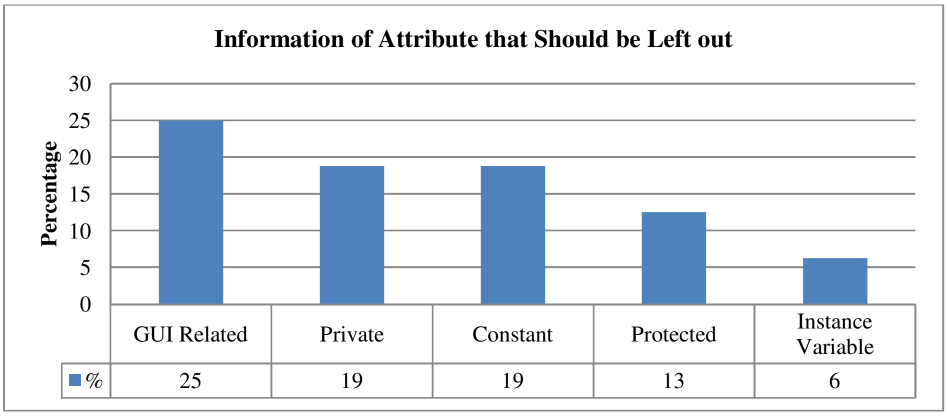


Figure 26: Information of Attribute that Should be Left out

Figure 26 shows the results of the Attribute category. Out of 6 elements from all subcategories, only five elements are shown in Figure 26 because there were no respondents that chose to exclude Public attributes in a class diagram. 25% of the respondents chose not to include the GUI related attributes. This information seems not important and based on our informal interview the respondents were more concerned on classes that are created by the programmer or software designer. 19% of the respondents like to leave out Private and Constant types of attributes. It follows with 13% of respondents that proposes to leave out Protected attributes. 3 out of 32 respondents (9%) think that all attributes should be left out from class diagrams. These respondents commented they only need class names and relationships in a class diagram.

4.2.2.2 Category 2 : Operation

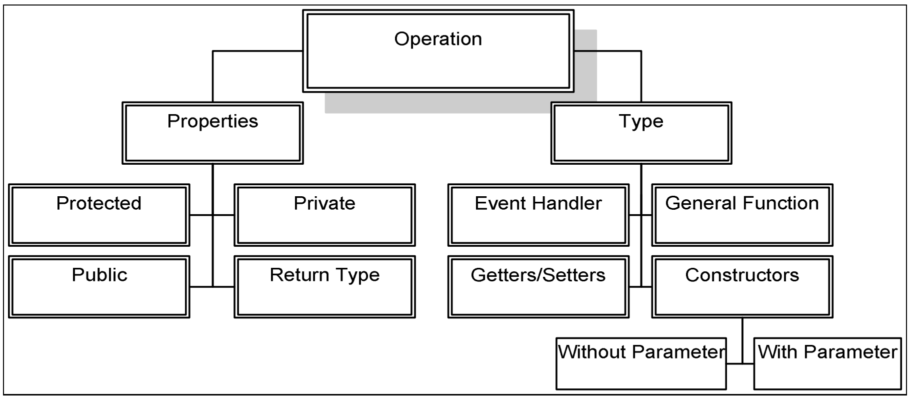


Figure 27: Operation Category

The Operation category is divided into two subcategories namely Properties and Type (Figure 27). The Properties subcategory consists of four elements and these are Private, Protected, Public and Return Type. The Type subcategory also consists of four elements and these are Event Handler, General Function, Getters/Setters and Constructor. The element Event Handler is an operation that handles events such as "addItemButton_Clicked" and "onButton1_Clicked". General Function is a function that is commonly used such as "toString()". Getters/Setters are operations that are used to access and update a variable in a class. In this study, we consider an operation as a getter or setter if the function uses the word 'get' or 'set' in the beginning of the name of a function e.g. getName, setCounter. For the Constructor element, we divided this element into 2 groups and these are With Parameter and Without Parameter because the respondents seem to differentiate this information.

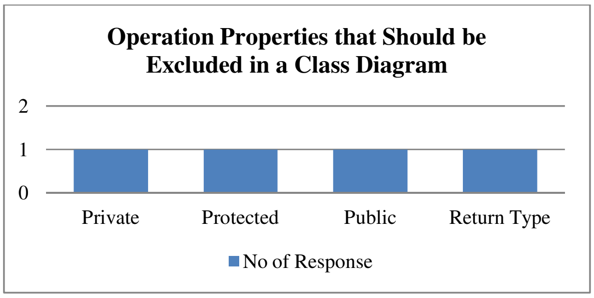


Figure 28: Operation Properties

Figure 28 shows the Operation Properties that have been chosen by the respondents to be excluded in a class diagram. It shows that there is only one respondent for every operation property that have chosen these elements that should not be included in a class diagram. The results show that the majority of the respondents have chosen that all elements in Operation Properties should be included in a class diagram.

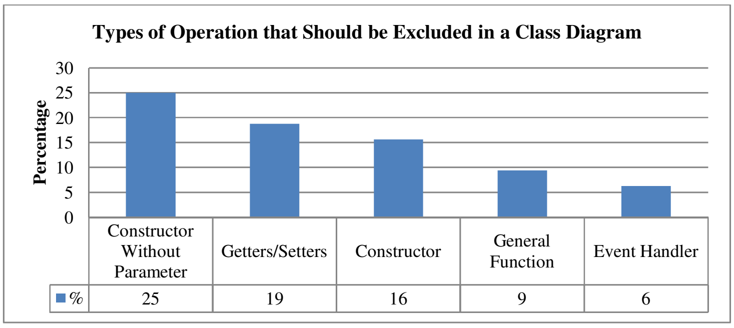


Figure 29: Type of Operation that Should be excluded in a Class Diagram

The results of the Type of Operation category are presented in Figure 29. The results show that 25% of the respondents chose to exclude Constructors Without Parameters. This type of operation is not important because it does not indicate any important information because the default initialization of an object is without parameters. Nevertheless, 16% of the respondents suggested that all Constructors should be left out in a class diagram. For Getters and Setters, 19% of the respondents suggested that these operations should be excluded in a class diagram. A reason for this could be that it is a common operation that is created for accessing and modifying variables in a class diagram. 9% of the respondents mentioned that General Functions should not be included in a class diagram because these functions are commonly used and well-known to programmers. Event Handlers were chosen to be excluded from a class diagram by 6% of the respondents. Most of the event handlers in the class diagrams in this survey are derived from GUI libraries. Apart of the result presented in Figure 29, 15% of the respondents indicated that all operations should be excluded from a class diagram. These respondents mentioned that only class names and relationships are needed in a class diagram.

4.2.2.3 Category 3: Class

As shown in Figure 30, the Class category is divided into two subcategories which are Type of Class and Role. The Type of Class subcategory consists of Interface, Enumeration, and Abstract elements while the Role subcategory consists of five elements which are Console, Listener, Input/Support

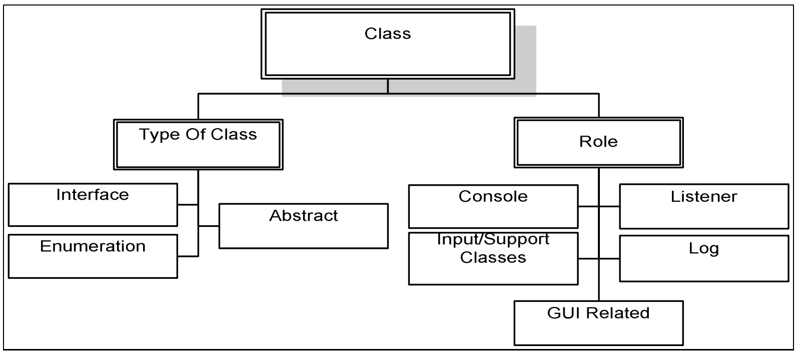


Figure 30: Class Category

Classes, Log, and GUI Related. The Role subcategory means that classes have specific role in a system. To exclude a class in a class diagram, we focus on the classes that perform a supporting role in a system such as GUI Related classes, Log classes and Console classes. For instance, class Log in the ATM system is categorized in the Role subcategory.

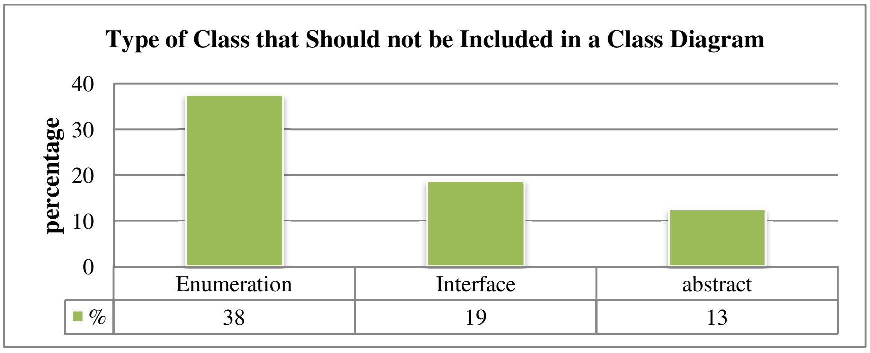


Figure 31: Type of Class that Should not be Included in a Class Diagram

For the subcategory Type of Class (Figure 31), 38% of the respondents chose not to include Enumeration classes. This is followed by Interface classes with 19% and 13% suggested that Abstract classes should not be included in simplified class diagrams. Enumeration classes are classes whose values are enumerated in the model as enumeration literals, which are not needed to understand a system.

Figure 32 shows the Role subcategory results. It shows that half of the respondents suggested that GUI related classes and classes for logging tasks

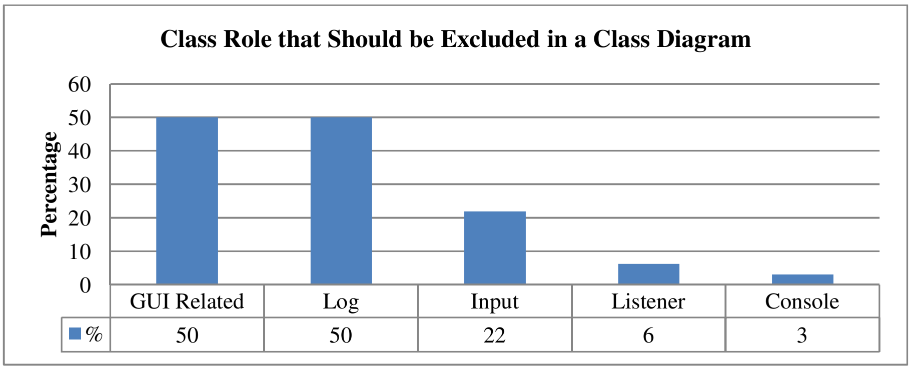


Figure 32: Class Role that Should be Excluded in a Class Diagram

should be left out in order to simplify a class diagram. Most GUI related classes were presented in the Library system and the Log class was presented in the ATM system. The respondents suggested eliminating these classes because without these classes you can still understand the system. The Input function is a class that is used to take the input from the interface or device. In the case of the ATM system, the “Money” and “Card” classes are an example of input function classes. 22% of the respondents said that this type of class should not to be included in a class diagram. The “Console” and “Listener” functions appear in the Pacman Game in Part B. These classes can be considered as classes that interact with the user input and other system input. There are 6% of the respondents that chose to exclude the listener function from the class diagram while 3% of the respondents chose to exclude the console function.

4.2.2.4 Category 4: Relationship

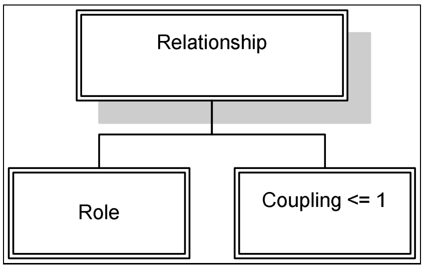


Figure 33: Relationship Category

The Relationship category is divided into two subcategories which are Role and Coupling <= 1. The Role subcategory means the role of the relationship that is labeled on this relationship. Coupling <= 1 means classes that have relationships equal to 1 or no relationship to other classes at all. Figure 33 shows the information about the subcategories for the Relationship category.

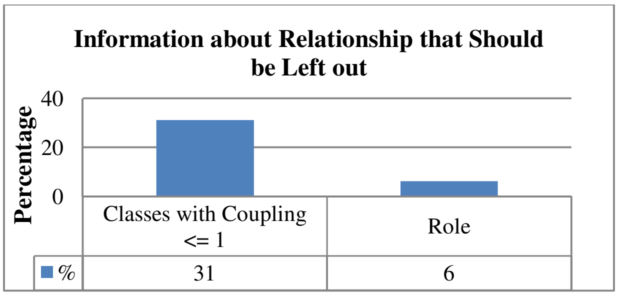


Figure 34: Information about Relationship that Should be Left out

Almost all the respondents that participated in this survey agreed that the Relationship element is important in a class diagram. However, there is some information related to the Relationship element that should not be included in a class diagram which are Classes with coupling less or equal to 1 and the Role of a relationship. 31% of the respondents intend to exclude classes with Coupling <= 1 because it seems that classes that only have coupling <= 1 are not important and more seen as a helper class. 6% of the respondents chose to remove the Role of relationship. The results are shown in Figure 34.

4.2.2.5 Category 5: Inheritance

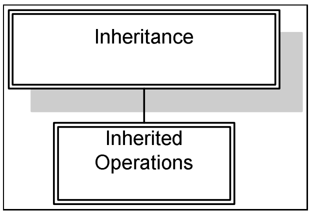


Figure 35: Inheritance Category

There is only one subcategory in the Inheritance category which is Inherited Operations. The Inherited Operations (see Table 14) are operations that are inherited from the parent class. This seems to be unnecessary to be presented in a class diagram. Figure 35 illustrates this subcategory of the Inheritance category.

Inheritance	**No of Respondents**
Inherited Operations	3

Table 14: Results of the Inheritance Category

4.2.2.6 Category 6: Package

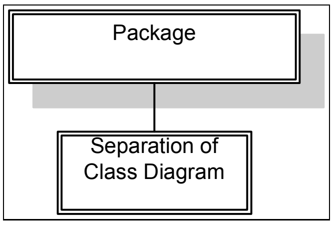


Figure 36: Package Category

In the Package category, there is only one subcategory which is Separation of Class Diagram. Figure 36 shows the structure of the Package category. This category was introduced because there were several respondents that separated the class diagram in such way that there were two class diagrams instead of one. Table 15 shows the results of the Package category.

Package	**No of Respondents**
Separation of Class Diagram	4

Table 15: Results of the Package Category

The amounts of classes in the three class diagrams are ranging from 15 to 22. Specifically in the Library System class diagram, there were 4 respondents that drew several lines to separate the GUI related classes from the classes that were created by the software developer. They suggested that the class diagram should be separated into two different diagrams. This basically means that they wanted to keep the GUI related classes and classes created for the system separated. There was one respondent that mentioned that the class diagram is too big and also there was one respondent that suggested that the class diagram should only consist of 5 to 7 classes in a class diagram. In Psychology there is a theory that humans can only focus on 7 ± 2 objects at the same time otherwise there are too many objects to focus on.

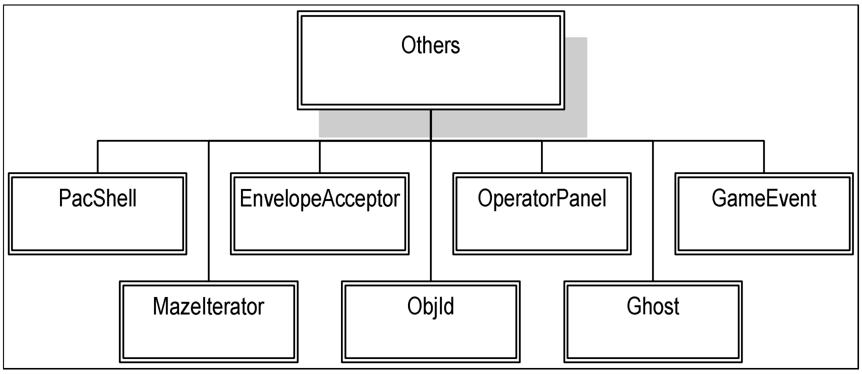
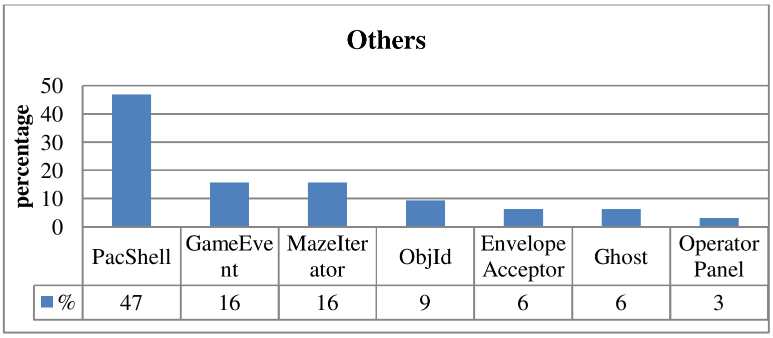


Figure 37: Others Category

4.2.2.7 Category 7: Others

Figure 37 shows several class names as a subcategory related to the “Others” category because we did not know in which other element they would fit in. PacShell, MazeIterator, Ghost, and GameEvent are in the Pacman class diagram. EnvelopeAcceptor and OperatorPanel are in the ATM System class diagram while the ObjId class is in the Library System class diagram.



	PacShell	GameEve nt	MazeIter ator	ObjId	Envelope Acceptor	Ghost	Operator Panel
■ %	47	16	16	9	6	6	3

Figure 38: Result of the Others Category

Figure 38 shows the overall results of this category. In this figure there are 46% of the respondents that chose to exclude the PacShell class from the Pacman Game. The PacShell class consists of the main function and this type of class is perhaps common in the programming language. 15% of the respondents excluded GameEvent and MazeIterator classes in the Pacman Game class diagram. The GameEvent class is excluded because the class looks like a helper class and it is only related to one class (i.e. coupling = 1). The MazeIterator class also looks like a helper class for another class (i.e helper class for the class Maze in the Pacman Game).

9% of the respondents excluded the ObjId class. A possible reason for this is that this class is a helper class that gives input to another class. Although this class has a lot of connections with other classes, it seems this helper class is not required to be shown in a class diagram. 6% of the respondents chose to exclude EnvelopeAcceptor and Ghost. The EnvelopeAcceptor class is only a helper class that is used to transfer data to another class. This could be the reason that the respondents decided to exclude this class. For the Ghost class, this may be a misinterpretation of the name of the class. There is a possibility that the respondents did not understand the role of the ghost in Pacman and they perhaps think that the Ghost class is a helper class or a dummy class. In fact, the Ghost class is an actor in the Pacman Game. 3% of the respondents chose to exclude the OperatorPanel class because the name is likely to present as a GUI related class.

Hence, from our analysis we may say that most of the classes that have been named in this category are either helper classes or GUI related classes. The respondents find such classes not important. This statement also becomes more valid if you look at Figure 40 in Part C (question C2A), which is described later.

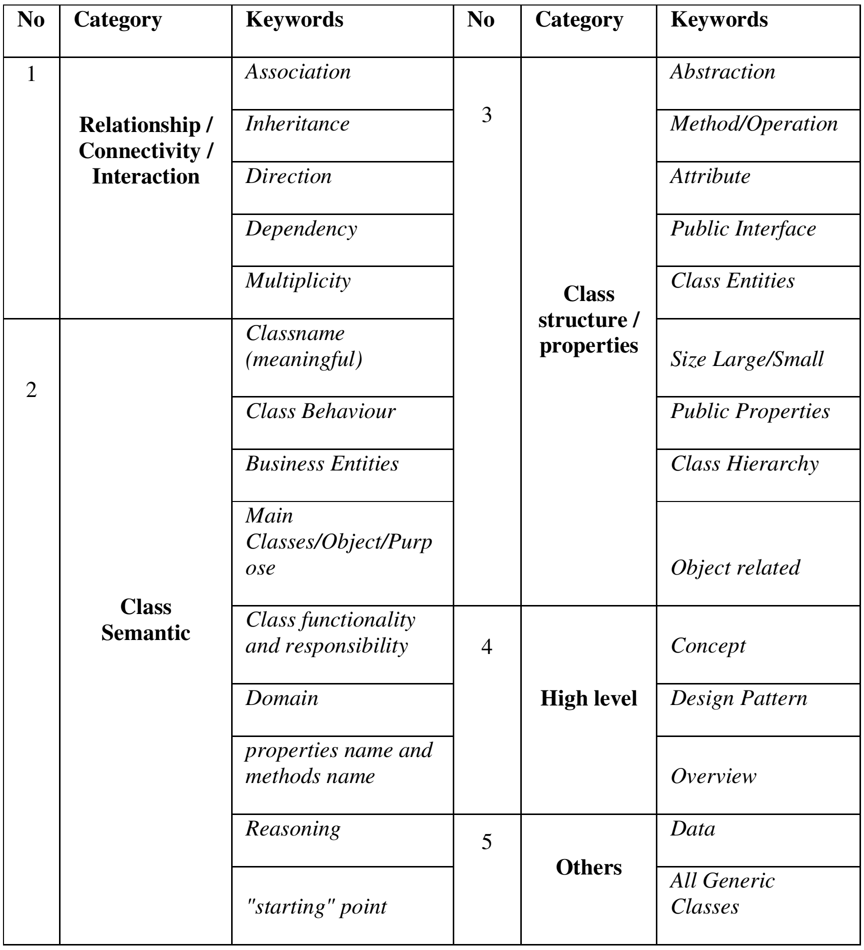
4.2.3 Part C: Class Diagram Indicators for Class Inclusion/Exclusion

Part C consists of six open-ended questions. The respondents were free to give any answer related to the questions. The answers given by the respondents were either precise or very broad. We initially observed the answers from the respondents and created several keywords to categorize these answers.

4.2.3.1 Question C1: In software documentation, particularly in class diagrams, what type of information do you look for to understand a software system?

In this question we were expecting to get the type of information that the respondents look for to understand a system. Based on an early observation of the given answers, we created several keywords and categorize those keywords to several categories. The keywords and categories are presented in Table 16. A detailed explanation of some of these keywords is the following:

1. **Relationship/Connectivity/Interaction:** The relationship between classes in a class diagram, which could be an association, inheritance, direction of the relation, dependency, and multiplicity.



No	Category	Keywords	No	Category	Keywords
1	**Relationship / Connectivity / Interaction**	*Association*	3	**Class structure / properties**	*Abstraction*
		Inheritance			*Method/Operation*
		Direction			*Attribute*
		Dependency			*Public Interface*
		Multiplicity			*Class Entities*
2	**Class Semantic**	*Classname (meaningful)*			*Size Large/Small*
		Class Behaviour			*Public Properties*
		Business Entities			*Class Hierarchy*
		Main Classes/Object/Purpose			*Object related*
		Class functionality and responsibility	4	**High level**	*Concept*
		Domain			*Design Pattern*
		properties name and methods name			*Overview*
		Reasoning	5	**Others**	*Data*
		"starting" point			*All Generic Classes*

Table 16: Keywords on Types of Information to Understand a System

2. **Semantic:** The semantic role of a class. This could be:

   (a) The **Classname** that should be meaningful to understand the class.

   (b) **Class behavior** is about the behavior of the class: what does the class do?

   (c) **Business entities** and **Domain** are about the classes that have some kind of business value or are meaningful for the domain of a class diagram, respectively.

   (d) **Reasoning** is about what kind of reason this class is in the class diagram.

   (e) The rest of the keywords in **Semantic Role** are straightforward.

3. **Class structure:** Types of information about the structural design of a class.

   (a) It could consist of **Methods** and **Attributes**.

   (b) It could also be an **Abstraction** or a **Public Interface**.

   (c) **Size large/small** is about the size of a class.

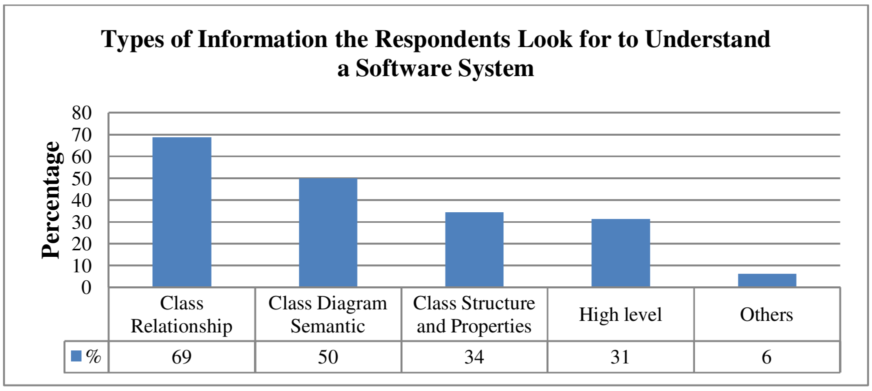
   (d) The rest of the keywords in **Class Structure** are straightforward.

4. **High Level:** In other words: High level of detail. This basically means that the class diagram must be as detailed as possible in which there are a **Concept** of the class diagram and an **Overview** of it. The class diagram could also contain **Design Patterns**.

5. **Others:** This group contains keywords we could not relate to the other group but that the respondents did say in this question.

   (a) Some respondents said that a class diagram should show what kind of **Data** a system needs.

   (b) Some respondents said that they only need **Generic Classes** to understand a system.



	Class Relationship	Class Diagram Semantic	Class Structure and Properties	High level	Others
■ %	69	50	34	31	6

Figure 39: Type of Information Look for in a Class Diagram

The results of this question are shown in Figure 39. The results obviously show that class relationship is the most important information in a class diagram that the respondent searches for understanding a class diagram. 68% of the respondents mentioned this. 50% of the respondents search for class semantics such as meaningful class names, class functionality and behavior, class properties and so on. It is possible that the respondents were trying to understand the structure of the system by searching for the classes that are related to the software domain. This means that class diagrams that can present semantics of classes (such as a good class name and class properties) would provide better software design understanding. About 34% of the respondents were looking at class properties such as attributes, operations, class interfaces and so on. This follows with 31% of the respondents that were looking at the class diagram high level abstraction for example design concepts, design patterns and class overviews.

From these results, it is obviously shown that the relationships between classes are important in a class diagram. It is the main information in a class diagram that most of the software developers look into. The semantics of a class such as meaningful classes, operations and attributes also play a major role to assist the software developer in understanding a system.

4.2.3.2 Question C2: In a class diagram, what type of information do you think can be left out without affecting your understanding of a system?

In this question we asked the respondents what type of information can be left out without affecting their understanding of a system. To make the answer a bit more specific we divided the question into four sections in which all sections must be answered. These four sections are: Classes, Operations, Relationships, and Other(s). In the first three sections we gave the participants some example answers that they could give as answers. In the last section they could add some other information that is not related to the first three sections.

A: Classes (for example: Helper class, Interface class, Library class, . . .)

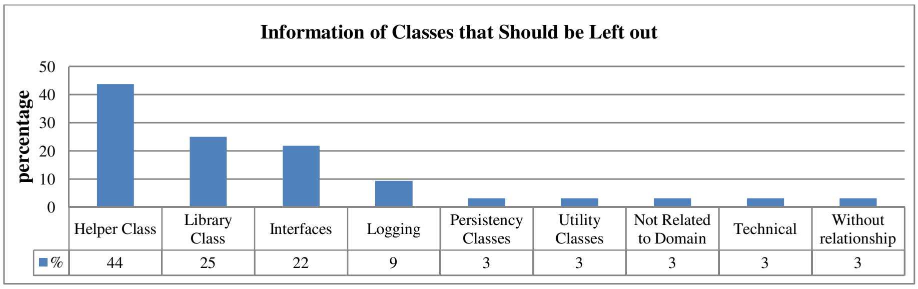
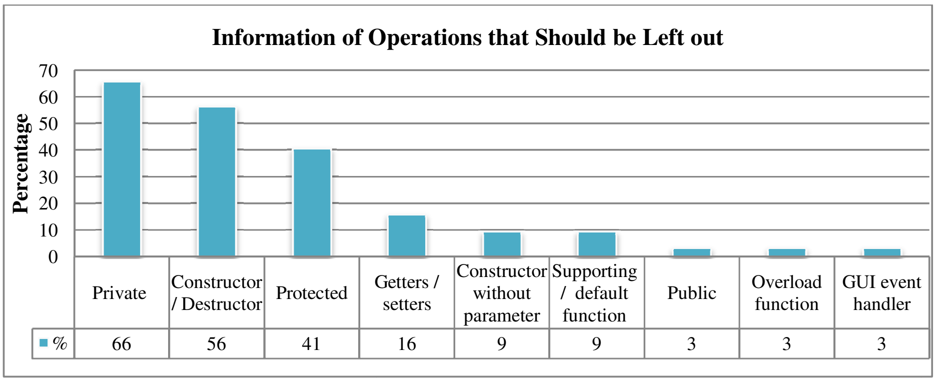


Figure 40: Information of Classes that should be left out

As shown in Figure 40, almost half of the respondents (44%) suggested that helper classes should not be included in a class diagram. However, it is not easy to detect a helper class in a class diagram. Detection based on the class name, operation, and attribute of the class may be used but it is not accurate since the helper class does not have a standard characteristic and it depends on the system domain and the software developer who exemplify it. This result also validates our result in the "Others" category in part B because most of those classes are helper classes as well.

A quarter of the respondents (25%) did not want library classes to appear in a class diagram. These library classes could make a class diagram more complex and hard to understand. The classes that should be included in a class diagram are only the ones that are created by the designer or programmer. 22% of the respondents suggested that the interface class type should not to be included in the class diagram. A reason for this could be that interface classes are GUI related and are not important for understanding a system. 9% of the respondents indicated that the log class should be excluded. The log class seems not important since it is common to create a log for a transaction or activity in a system. Also, log classes are normally linked or related to other classes that could make the class diagram more complex. Infrastructure, technical, framework, classes not related to domain, and classes without relationship were said to be left out from a class diagram by 3% of the total respondents.

B: Operations (for example: private, protected, public, constructor, . . .)



	Private	Constructor / Destructor	Protected	Getters / setters	Constructor without parameter	Supporting / default function	Public	Overload function	GUI event handler
■ %	66	56	41	16	9	9	3	3	3

Figure 41: Information of Operations that should be left out

Figure 41 shows that 65% of the respondents chose to exclude private operations in a class diagram. Constructors and destructors are also types of operations that are not needed in a class diagram (56% of the respondents) in order to understand a system while only 9% of the respondents say that they do not need constructors without parameters. 40% of the respondents mentioned that protected operations should be left out from a class diagram. A reason for this could be that this type of operation can be assumed as a private operation but appears public to some classes only. It was quite a surprise that not many respondents suggested to remove mutator methods (getters/setters) from the class diagram since these operations can be integrated in other operations that a system actually needs. Supporting/default functions such as "toString" should be excluded from a class diagram mentioned by 9% of the respondents. 3% of the respondents suggested that public operations, overload functions and GUI event handlers should be left out.

It was quite surprising to discover that most of the respondents suggested excluding private operations in a class diagram because from the results in Part B, only 1 respondent mentioned about this. Hence, this is kind of contradicting but a reason for this could be that the diagrams that we have given to our respondents do not provide many private operations which means that there is a possibility that they have not seen these private operations in these class diagrams.

C: Relationships (for example: labels, multiplicities, self-relations)

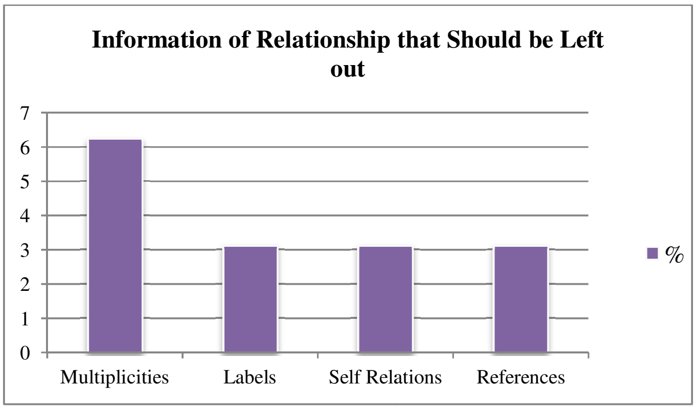


Figure 42: Information of Relationship that should be left out

Multiplicity is what most respondents mentioned that is not needed in a class diagram (see Figure 42). However, only 6% of the respondent mentioned this, which is a quite low percentage. 3% of the respondents do not need any labels (or roles of the relationships), self relations and references in a class diagram.

D: Other(s)

As shown in Figure 43, about 9% of the respondents said that private fields should not be included in a class diagram. Only 3% of the respondents suggested technical, duplicates and UI information not to be included in a

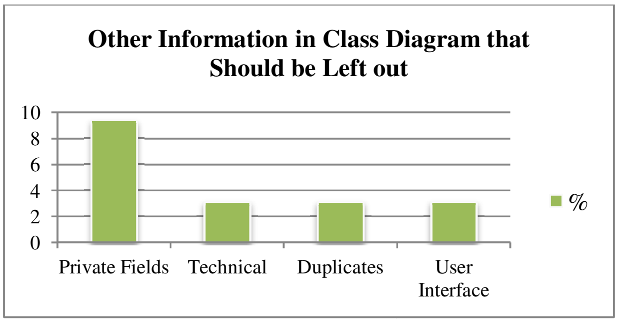


Figure 43: Other Information in Class Diagram that should be left out

class diagram.

4.2.3.3 Question C3: Do you think that a class diagram should show the full hierarchy of inheritance? If not, which parts could be left out? (for example: parent, child, intermediate parent/child, leaf, . . .)

This question aimed to discover whether inheritance in a class diagram should be shown in full hierarchy or not and if not, which part of the inheritance information should be left out.

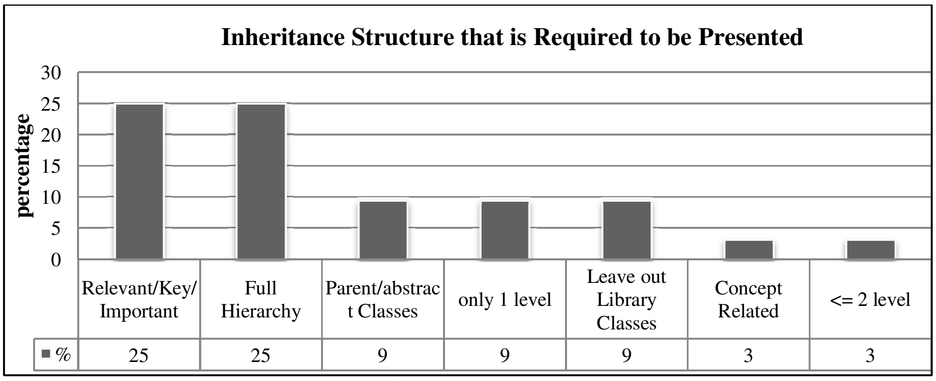


Figure 44: Inheritance structure that is required to be presented

Figure 44 shows that 25% of the respondents suggested that only relevant/key/important classes should be presented in a class diagram regardless if it is in an inheritance hierarchy while another quarter of the respondents want the full hierarchy of the inheritance to be shown in a class diagram (answered 'Yes'). 9% of the respondents only want the parent class to appear in the class diagram and also 9% of the respondents suggested that the hierarchy should only consist of 1 level of children or parents. 3% of the respondents mentioned that they only need inheritance that is concept related and another 3% mentioned that a maximum hierarchy level of 2 should be enough.

This result shows that the respondents needs full hierarchy, but if the classes can be identified as key/important/relevant classes, the inheritance can be simplified by only showing these key/important/relevant classes.

4.2.3.4 Question C4: What criteria do you think indicate that a class (in a class diagram) is important for understanding a system?

In this question we try to discover what criteria indicate that a class in a class diagram is important for understanding a system.

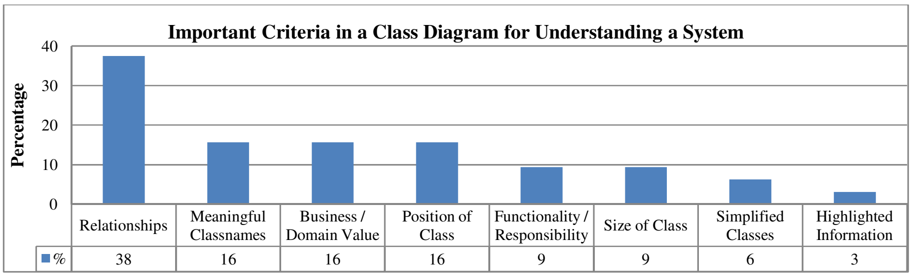


Figure 45: Important criteria in a Class Diagram for Understanding a System

As the results in Figure 45 show, 38% of respondents think that relationship is the most important criterion in a class diagram which also validates our results in question C1. 16% of the respondents think that meaningful class names, business or domain value, and the position of class are the important criteria in a class diagram. This result shows that semantics and the meaning of a class play a role in understanding a class diagram. Some respondents preferred to search for the position of the class and most of them mentioned that the middle of a class diagram should contain the important classes. 9% of the respondents mentioned that the size of a class determines if a class is important or not which is structural related. Another 9% said that the functionality/responsibility of a class is important which is again semantic related. Simplified classes and highlighted information were used as answers by 6% and 3% of the respondents, respectively. These results are aligned with the results for question C1 where the important criteria in a class diagram are relationship and semantic information.

4.2.3.5 Question C5: If you try to understand a class diagram, which relationships do you look at first? (Example: dependencies, inheritance, associations, etc)

This question aims to find out the type of relationship the respondents look at first to understand a class diagram. Three types of relationships were provided as example answers. The answers are quite biased because most of the answers only mentioned about these types of relationships. None of the respondents answered other types of relationship such as composition, aggregation, and realization.

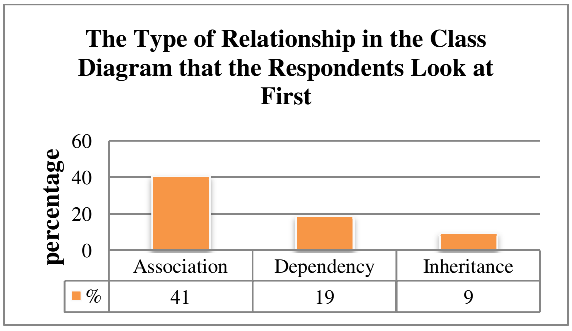


Figure 46: The Type of Relationship in a Class Diagram that the Respondents Look at First

Figure 46 shows the results of this question. It shows that 41% of the respondents liked to search for association relationships first while 19% search for dependency relationships. Only 9% of the respondents search for inheritance relationships. Several respondents answered with more than one relationship and some respondents ordered these three relationships in an order of importance. We only took the first answer given so if someone answered “association and dependency” for example then we only took the first relationship which is association in this example. This result obviously shows that the association relationship is the most important relationship in a class diagram. The association relationship is important to show the relationship between classes.

4.2.3.6 Question C6: If there is a tool for simplifying class diagrams (e.g. obtained from reverse engineering), what features/functions would you expect from such a tool?

In this question, we try to discover what kind of features the respondents are looking for if there is a tool which could simplify class diagrams.

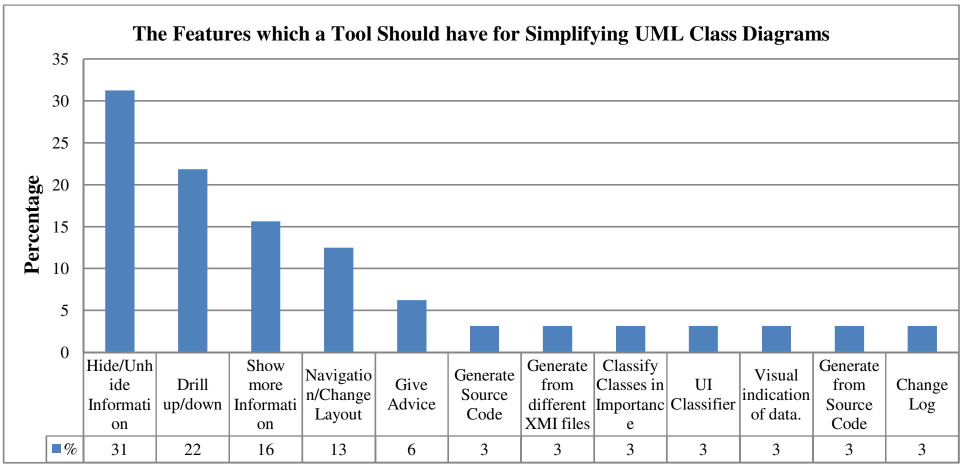


Figure 47: The Features that a Tool Should have for Simplifying UML Class Diagrams

Figure 47 shows the results of this question. The results show that the respondents mainly want a tool that can hide/unhide information (31% of the respondents). The other feature that relates to this is the drill up/down feature because when you are drilling up, the amount of information of a class diagram will be less and vice versa. 22% of the respondents want such a feature. These two features are different from each other because in the first feature you can manually mark the information that you do not want and the second feature automatically lessens the information when you are zooming out.

16% of the respondents want to see more information about a class by hovering over a class in a class diagram for example. This could show you different kinds of information, for example the amount of relationships this class has. Another feature that many respondents want (13% of the respondents) is the changeable layout of the class diagram in which the navigation can be improved. An option could be to resize the layout of the class diagram. 6% of the respondents want to have a feature in the tool that can give advice that could improve the class diagram. The other features that are shown in this figure are being desired by only 1 respondent (3%) each.

Thus, these results show that the respondents want a tool in which the user of this tool can manipulate the level of detail of a class diagram by hiding/showing information or zooming out/in of the class diagram.

4.3 Discussion

In this section we discuss the results and findings presented in the previous section. The discussion is divided into five subsections: Respondents' Background, Class Properties, Class Role and Semantics, Class Diagram Simplification Tool Features, and Threat of Validity.

4.3.1 Respondents' Background

In Part A, we have accessed the information about the respondents' skills and experiences in UML, particularly in class diagrams. Most of the roles of the respondents were programmer and half of them are software architects. Although there were respondents that are project managers, they were also a programmer. This shows that the respondents of this survey were directly involved with the software development process. None of the respondents in this survey chose the answer 'No' for question A3 which means all of the respondents have knowledge in UML. 75% of the respondents have more than 5 years experience in class diagrams where 50% of the respondents have experience over more than 10 years. In terms of learning UML, most of the respondents that have more than 10 years experience learn UML by themselves.

In terms of the respondent's skill in class diagrams, we found that 88% of the respondents have at least average skill in creating, modifying and understanding class diagrams. Then, we tried to discover the artefacts that the respondents prefer for understanding a system. The artefacts are source code and UML (class diagrams). The results show that there is no significant difference between the usage of source code and UML in order to understand a system. However, it was quite a surprise when we found that most of the software architects and software designers prefer source code rather than UML to understand a system. A reason for this result could be that the software architects and software designers have a good knowledge in programming or they have other techniques rather than UML for understanding a system. From our informal discussion, some of the respondents mentioned about sequence diagrams that are more helpful to show the behavior of the system and some of them use boxes and lines to show the component interaction of the system. Further research why these respondents do not prefer to use class diagrams should be done in order to get what type of information that is lacking in UML class diagrams.

4.3.2 Class Properties

Based on the results in Part B and Part C, it is not a surprise that we discovered that the most important element in a class diagram is class relationship. It is well-known that the relationship in a class diagram is an important element to show the structure of classes in a class diagram. Without relationships, a class diagram would only be a list of classes without showing which class is involved with the other. In terms of inheritance, a quarter of the respondents needs full hierarchy of the inheritance tree to be presented but another quarter of the respondents mentioned only classes that are relevant or important should be presented. Most of the respondents in this survey looked at association relationships first. This shows that the association relationship is important in class diagrams. However, this result is not really accurate since the respondents only gave the answer within the examples given in the questions.

In this survey, we found that most of the respondents suggested leaving out or separating the GUI related information from the class diagrams. In Part B we provided three class diagrams and they were required to eliminate information that is not important without affecting their understanding of the system. The results showed that most of the respondents suggested that the GUI related information should be excluded in a class diagram. From our observation, the respondents focuses more on the class diagrams' information (e.g. Attributes, Operations, Classes) that is created by the programmer or software designer. The GUI related information exists in source code (also appeared in reverse engineered design) when a developer used GUI libraries provided by Rapid Application Development (RAD) tools such as JBuilder, Delphi and Visual Basic. The Library system's class diagram shows a lot of GUI Information and many respondents suggested this information should be left out. Also, there were several respondents that drew a border between the software developer created classes and classes that were derived from the GUI library.

In terms of class operations, most of the respondents suggested to leave out the private and protected type of operations. These types of operations are only used for internal classes and member classes for protected operations. It seems not to be important because they could not be accessed publicly from other classes. We also discovered that constructor/destructor operations should not appear in simplified class diagrams. Particularly in Part B, we found that most of the respondents suggested that constructors without parameters should not be presented in a class diagram. Mostly, the constructor is a default function provided by development tools when a class is created. It is not important to have default constructors (without parameter) information in a class diagram but the constructors that have parameters are crucial to be presented in a class diagram because it is used for object initialization purposes.

4.3.3 Class Role and Semantics

One of our useful discoveries in this study is the importance of the class role and semantics in a class diagram. Class roles based on class name are important because from our observation the respondents seemed to try to understand a system based on class name and role. Not only class names can present a role of a class, the operation name and attribute name are also crucial. This showed when we asked about the important criteria in a class diagram for understanding a system, most of the respondents mentioned 'meaningful' class names, business and domain value related and also functionality or responsibility. From these meaningful class names, they tried to understand the structure and also interact between classes in a system. By using this information, they can get an overall idea on how a system works and get some hints of the functionalities of classes in a class diagram.

In this survey we also discovered that classes that should be left out in a class diagram are helper classes, library classes and interfaces classes. Most of the respondents suggested leaving out helper classes. Nevertheless, it is not easy to automatically identify helper classes based on the class name or other information because it only can be identified manually by the software developer and the results are different based on the software developer's experience. Helper classes could possibly be detected if the criteria of the helper classes would be available.

4.3.4 Class Diagram Simplification Tool Features

From this survey, we found out that most of the respondents need tools for simplifying a class diagram that can hide/unhide information and drill up and down a class diagram. These features are needed most because they are able to zoom in and zoom out in a class diagram. With this feature they can use the tools to understand the system in general by leaving out the details and they can get more information when they want to modify the system. From our informal discussion with the respondents, simplification of class diagram is needed when they want to understand the overall system design but detailed information in class diagrams is needed in modification tasks. Hence, both simplified and detailed designs are needed.

4.3.5 Threat of Validity

The statement of the questionnaire in the introduction "The task focuses on software design comprehension after the software implementation phase or during software maintenance phase" is a too broad area. The respondent may get confused that the class diagram is constructed after the maintenance (for documentation purpose) or the class diagram is constructed before the maintenance phase. Also, in Part A, the respondents may get confused whether to choose software architect or software designer because both roles were used in different terms but with the same meaning.

4.4 Conclusion

This study presented a survey on how to simplify a class diagram without affecting their understanding of a system. In particular, the questions in this survey were about what information should be left out from a class diagram and also what kind of important information should remain. 32 software developers from the Netherlands participated in this survey.

From the results, it is not a surprise that the most important element in a class diagram is the relationship. Class relationship is important to show the structure of a system. The type of relationship that the developers look at first is the association and dependency after. In this survey we discovered that the class diagram's role and semantics are important because most of the respondents search for meaningful class names and class roles in order to get high-level understanding on how a system works. This means, meaningful class names, operation names and attribute names are important to show the functionality or responsibility of a system.

To simplify a class diagram, most of the respondents chose to exclude GUI related information and also library classes. This shows that most of the software developers need the information about the classes that are created or designed, but not the classes that are generated or commonly used (e.g. Library class). Most of the respondents also mentioned that helper classes should be excluded to simplify a diagram however it is not easy to automatically identify a helper class. Private operations, protected operations and constructors (without parameter) are types of operations that should be left out in order to simplify a class diagram. These types of operations seem not to be important. Although we are aware that research on validation of our approach needs to be done, we found several useful indicators that could be used in the future for class diagram simplification.

5 Conclusions

In this chapter we present a summary of our findings of this study. Then, we present several recommendations based on our analysis. Next, we describe some future works that can be done after this study. Finally, we present our conclusions based on the analysis of these two questionnaires.

5.1 Summary of Findings

In this section we present a summary of our findings and present the similarities and contradictions of the given responses between the two questionnaires.

In part A, we asked the respondents in both questionnaires about their background. Some questions between these two questionnaires were, however, different. We discovered that 88% of the respondents in the non-structural questionnaire had rated themselves average or above for their skills in creating, modifying and understanding class diagrams while for the structural questionnaire, 76% of the respondents rated themselves average or above. In term of percentage, there was not much difference but for years of experience the respondents of the non-structural questionnaire had higher years of experience than the respondents of the structural questionnaire.

In part B and C of the non-structural questionnaire, we have discovered that relationship is the most important element to show the structure of the classes in a class diagram. This can also be validated if we look at part C in the structural questionnaire. Coupling was a very influential factor for the respondents when it comes to excluding classes from a class diagram. Most of the times, classes were excluded when the coupling was <= 2. This means that if a class contains many relations then this class is important. Also, the Dep_Out and Dep_In metrics were one of the highest scores in the coupling category, which further validates this discovery.

In terms of inheritance in the structural questionnaire, we discovered that if a class contains a high number of children, then this class is important. This metric scored 20 points and was the highest in the inheritance category. In the non-structural questionnaire, we discovered that one quarter wants the full hierarchy and another quarter only wants the relevant classes. This last result validates the result of the structured questionnaire since DIT and CLD are one of the lowest scoring metrics and the respondents thus do not need the full hierarchy.

In the non-structural questionnaire we also discovered that classes that GUI related should be excluded since the respondents were more focused on the information that was created by the programmer or software designer. This discovery was also found in the online questionnaire. Most of the respondents excluded classes like “AboutDialog”, “MessageBox”, and “QuitDialog” in the Library system, which are GUI related classes. Also interface classes were excluded by most of the respondents in the Pacman class diagram.

In terms of operations, we discovered that in both questionnaires the respondents preferred public operations since these operations are not restricted to one class. In the structured questionnaire, the metric that counts the number of public operations scored 25 points and was also the most important metric based on the results. In the non-structural questionnaire, we discovered that most of the respondent suggested leaving out the private and protected operations, hence our statement that the respondents preferred public operations. Other discoveries in the non-structural questionnaire are that respondents also suggested leaving out the constructors, specifically constructors without parameters. This is because the constructor is a default function provided by development tools when a class is created.

One of our useful discoveries in this study is the importance of the class names. We have found out in the non-structural questionnaire that the respondents seemed to try to understand a system based on the class names and roles. This was shown when we asked the important criteria in a class diagram for understanding a system. Many respondents mentioned that there must a story around the class diagram and should show the functionality and flow of the system. Most of respondents mentioned that a class diagram needs to contain "meaningful" class names and must be domain related. This can be further validated by inspecting the structured questionnaire. In question 14 of part B we discovered that the keyword "Domain related" was one of the most important keywords in this question that the respondents mentioned. Also, the class names were an influential factor in part C of the structured questionnaire.

Another useful discovery is that the respondents preferred the reverse engineered class diagram in the structured questionnaire. However, the source code must correspond with its forward design in order to get a good reverse engineered class diagram. Thus, this is the reason why the respondents preferred the reverse engineered design of Pacman over the reverse engineered design of the Library system because the Library system might not correspond to its forward design.

Another discovery in the non-structural questionnaire was that most of the respondents suggested leaving out helper classes. Nevertheless, it is not easy to automatically identify a class as a helper class. As for the class diagram simplification tool features, most of the respondents wanted a feature to manually hide/unhide information in a class diagram. Another feature many respondents desired is to zoom in and out of the class diagram so that the class diagram shows more or less information.

5.2 Recommendations

We recommend highlighting the classes in a class diagram based on our analysis. These highlighted classes can be used to advise the software developer/maintainer as a hint on which classes should be included or excluded in a simplified class diagram.

We also recommend that all classes in a class diagram should have a meaningful name that can present the functionality or the features that is provided by the class. This also applies with the names of the attributes and operations. By using meaningful names, the software developer can understand better and faster because they can predict the flow of the system and the class' functionality.

Coupling is also a very important element in a class diagram. Hence, it is advisable to focus on classes that have a high amount of coupling. These classes may present that the class is important. This study is about simplifying class diagrams. However, this information can be used for the software developer on how to read a class diagram based on the important elements.

5.3 Future Works

This study was an early experiment on how to simplify a class diagram and we see a number of ways to extend this work. In Part B of the non-structural questionnaire, we have used the reverse engineered class diagram and forward engineered class diagram in two separate groups. Also, we have used the different Levels of Detail in different sets of groups. The comparison of these different flavors of class diagrams in terms of what information the respondents suggest to leave out can be the future work to extend this study. It would be interesting to compare the results between these class diagrams and see if there are any differences in what the software developers are excluding from these diagrams. Besides this, it would be interesting to further research the preferences of the respondents between the forward design and the reverse engineered design.

Also from this study, we have discovered information that should be left out to simplify a class diagram and what metrics are important. By using this information, a simplified class diagram could be produced. 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. This proposed study may discover other information that is needed in a class diagram and other information that can be excluded. It would also be interesting to include other metrics that we have not chosen and check whether they are important or not.

From the results, we found that class role and responsibility are one of the important indicators in a class diagram. The role and responsibility of a class are detected by using the class names, operation names and attribute names. 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. The results of this study can be used to predict the important classes in a class diagram.

Our last suggestion is about the two questionnaires itself. We discovered some flaws in both questionnaires and our suggestion is to make these two questionnaires better by adding questions that we have not asked yet or changing the current questions to better understandable questions. Also, our amount of responses, especially in the online survey, can be considered as low. It would be interesting to see what the results are with a larger group of respondents.

5.4 Conclusions

In this study we have created two different questionnaires to find out what kind of information should be left out and what metrics are important in a class diagram in order to simplify a class diagram. One of the two questionnaires was created online and we received 25 complete responses. In the other questionnaire, 32 software developers from The Netherlands participated in this survey. We have discovered the important elements that should be included in a class diagram.

We discovered that the relationship is one of the most important elements in a class diagram, which is obvious and also not a huge surprise. This has been an important element in both of the questionnaires. We also discovered that class names are one of the most influential factors when trying to exclude classes from a class diagram. This means that the class diagram's meaningful names that represent the classes' role and semantics are important. In this study we also discovered that public operations are preferred by the respondents. The NumPubOps metric was the highest scoring metric and in the other questionnaire, most of the respondents mentioned that they do not prefer private and protected operations. Also, constructors, especially constructors without parameters, should be excluded in a class diagram.

Most of the respondents also mentioned that they would exclude GUI related information and also library classes. This basically means that the software developers only want classes that are created by the designer. Helper classes and the type of classes should be omitted in a class diagram. However, it is not easy to detect a helper class.

In this study we discovered that respondents preferred the reverse engineered class diagram over the forward designed class diagram. However, these reverse engineered designs must correspond with its source code that implemented the forward design. By doing this, a reverse engineered class diagram will be created that is similar to the forward design or even better. In terms of inheritance, many respondents only need the important/relevant/key classes in the hierarchy. We also found out that if a class has a high number of children, then this class is important. Overall, based on our results, we have found various important elements that should be included in a class diagram. We hope to see in the future that the class diagrams are created in such way that it is easier for the software developers to understand what the system does. We hope that we have contributed some important information to achieve this goal.

References

[1] Enterprise Architect. `http://www.sparxsystems.com.au/`.

[2] Limeservice. `https://www.limeservice.com/en/`.

[3] Research Methodology: An Introduction. `http://www.newagepublishers.com/samplechapter/000896.pdf`.

[4] SDMetrics. `http://www.sdmetrics.com/`.

[5] Wikipedia: Unified Modeling Language. `http://en.wikipedia.org/wiki/Unified_Modeling_Language`.

[6] S. Bassil and R. K. Keller. *Software visualization tools: Survey and analysis*, volume 67, pages 7–17. IEEE, 2001.

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

[8] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on uml class diagrams. *Artificial Intelligence*, 168(1–2):70–118, 2005.

[9] R. C. Bjork. Atm system. `http://www.math-cs.gordon.edu/courses/cs211/ATMExample/`.

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

[11] A. Craig, A. Dinardo, and R. Gillespie. Pacman game. http://code.google.com/p/tb-pacman/.

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

[13] H.-E. Eriksson, M. Penker, B. Lyons, and D. Fado. *UML 2 Toolkit*. Wiley, 2004.

[14] A. M. Fernández-Sáez, M. Genero, M. R. V. Chaudron, and I. Ramos. *A Controlled Experiment on the Impact of UML Diagram Origin on Maintenance Performance*. Submitted for publication.

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

[16] R. Koschke. Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. *Journal of Software Maintenance and Evolution Research and Practice*, 15(2):87–109, 2003.

[17] F. Leemhuis. Devnology community. http://devnology.nl/.

[18] A. Nugroho and M. R. V. Chaudron. *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, September 20-21, 2007.

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

[20] H. Osman and A. van Zadelhoff. Non-structured questionnaire. http://www.liacs.nl/~hosman/Questionnaire.rar.

[21] H. Osman and A. van Zadelhoff. Non-structured questionnaire responses. http://www.liacs.nl/~hosman/SurveyData.rar.

[22] H. Osman and A. van Zadelhoff. Structured questionnaire. http://www.liacs.nl/~hosman/The_Presence_of_Classes_in_Class_Diagrams.pdf.

[23] H. Osman and A. van Zadelhoff. Structured questionnaire responses. http://www.liacs.nl/~hosman/Complete_Results_Structural_Survey.rar.

[24] A. Parasuraman. *Marketing Research.* Addison-Wesley Publishing Company, second edition, 1991. `http://www.sciencebuddies.org/science-fair-projects/project_ideas/Soc_survey.shtml`.

[25] S. Yusuf, H. Kagdi, and J. I. Maletic. Assessing the comprehension of uml class diagrams via eye tracking. *15th IEEE International Conference on Program Comprehension ICPC 07*, pages 113–122, 2007.