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

Osman, M.H.B.

Chapter 6

Exploring the Suitability of Object-Oriented Design Metrics as Features for Class Diagram Simplification

Class diagrams may include an overwhelming amount of information. For large and complex class diagrams, there is a possibility that not all information in the class diagram is important for understanding the system. In this chapter, we study how to identify essential and secondary information in class diagrams. To this end, we performed a survey with professionals, academics and students to enquire how to decide which information in class diagrams is considered important. In particular, we explore whether software design metrics can be used as a means of identifying important classes in a class diagram.

In total, 25 complete responses were received. 76% of the respondents have average or above skills with class diagrams. We discovered that the metric that counts the number of public operations is the most important metric for indicating the importance of a class in a diagram. Also, we discovered that class names and coupling were influencing factors when it comes to excluding classes from a class diagram.

This chapter is a more detailed version of a publication entitled **"UML Class Diagram Simplification - A Survey for Improving Reverse Engineered Class Diagram Comprehension"**,In Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development(MODELSWARD 2013)

6.1 Introduction

The UML class diagram is one of the valuable artifacts 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 software systems. UML class diagrams describe the static structure of programs at a higher level of abstraction than source code [70].

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 [41]. With this technique, recovery of a class diagram can be done based on the source code. However, the resulting class diagrams from reverse engineering techniques often suffers from too much detail and information. In particular, RE-CDs 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 [126]. 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 [59], 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, the information that is needed by developers or maintainers to be shown in a class diagram should be discovered.

In this chapter, we aim at simplifying UML class diagrams by leaving out unnecessary information without affecting the developer's understanding of the system. Based on the feedback in Chapter 5, the software developers indicated that the system structural information (e.g. relationship and class elements) influence the determination of classes that could be included (inclusion) and classes that could be omitted (exclusion) in the class diagrams. To this end, we have conducted a survey to gather information from IT professionals, researchers or academics and students about what type of information they focus on. The survey directed to the structural information based on the object-oriented design metrics (i.e. the metrics from size category, inheritance category and coupling category). We prepared a questionnaire that consisted of 24 questions that are divided into 3 parts in order to discover this information.

The chapter is structured as follows. Section 6.2 discusses related work. Section 6.3 describes the properties and tools for this research. Section 6.4 explains about the survey methodology while Section 6.5 presents the results and findings. We discuss our findings in Section 6.6. This is followed by our conclusions in Section 6.7 and future work in Section 6.8.

6.2 Related Work

In this section, we discuss several studies that are related to the research in this chapter.

6.2.1 Usage of design metrics

Design metrics have been used for various purposes in software engineering. The Chidamber and Kemerer metrics (widely known as CK metrics) are well-known object-oriented metrics that is commonly used in software maintenance and fault proneness research.

In software maintenance, Li et al. [104] use object-oriented metrics (CK) to predict software maintenance effort. Their study shows a strong relationship between maintenance effort and object-oriented metrics. Their study also validates the results to prove the ability of these metrics to predict the maintenance effort. Binkley et al. [27] investigate more specific to coupling metrics in predicting maintenance measure. Their study found that the coupling dependency metrics (CDM) is suitable for predicting maintenance measures (e.g. a low level of coupling undergoes less maintenance and have fewer maintenance faults). They also found that CDM also suitable to predict the number of run-time failures.

There are several works on the usage of object-oriented design metrics associated with fault-proneness. Briand et al. [36] [35] found that several CK metrics were associated with the fault-proneness of classes. Also, Tang et al. [163] found that the Weighted Methods Per Class (WMC) and Response for a Class (RFC) metrics are associated with fault-proneness. El Emam et al. [53] found that inheritance and Export Coupling (EC) Metrics are associated with fault-proneness. In later research, Gyímothy et al. [73] discovered the Coupling between object (CBO) is the best metrics to predict fault-proneness.

In our study, we used several object-oriented metrics to predict the class should be included and should be excluded in the class diagram. We measure the influence of the object-oriented metrics in predicting class inclusion/exclusion by the score ranking based on the respondents' answers.

6.2.2 Automated Abstraction of Class Models

Falleri et al. [56] proposed an approach for class model normalization to produce a simplified class diagram by removing redundant information. The Relational concept analysis and Formal Concept analysis are used to process the normalization of the class model. Similar to our study, they also suggest that the usage of elements name for class model abstraction.

Egyed [51][52] proposed an approach for automated abstraction that allows designers to "zoom out" on class diagrams to investigate and reason about their bigger picture. This approach was based on a large number of abstraction rules. In total, the

article provides 121 rules to abstract a class diagram. However, this work is more concentrated on the abstraction of classes relationship. It could not automatically select the classes that should be included and should be excluded in class diagrams.

In our study, we conduct a survey to find out which object-oriented design metrics influences the software developer in selecting the classes that should be included and excluded.

6.3 Examined Properties and Tools

In this section, we describe i) the design metrics that we consider, and ii) the tools used for this research.

6.3.1 Examined Properties

SDMetrics [180] is an object-oriented design measurement tool for the Unified Modeling Language (UML). SDMetrics is capable of measuring 32 types of class diagram metrics which are divided into five categories, namely Size, Coupling, Inheritance, Complexity and Diagram. However, in this study, we only used 14 metrics from the categories of Size, Coupling and Inheritance. These 14 metrics are selected based on our initial experiments in Chapter 3. We reverse engineered the source codes of the case studies presented in Chapter 3 and extracted all object-oriented design metrics provided by SDMetrics. We found that only 14 metrics (as shown in Table 6.1) have significant value for measurement.

6.3.2 Tools

SDMetrics [180] is used to measure the structural properties of object oriented design. SDMetrics version 2.11 (academic license) is used for this purpose. We chose Enterprise Architect [153] version 7.5 for creating forward design and RE-CDs for this survey.

6.4 Survey Methodology

This subsection describes the design of the questionnaire. We explain how the questionnaire was designed and why. We also describe our online survey experiment that explains how the experiment was conducted.

6.4.1 Questionnaire Design

The questionnaire consisted of 3 parts i.e. part A, B and C. There was a total of 24 questions in this questionnaire.

Table 6.1: *The Chosen Software Design Metrics [180]*

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 in a class with a name starting with 'set'.
5	Getters	Size	The number of operations in a class with a name starting with 'get', 'is', or 'has'.
6	NOC	Inheritance	Number of Children (NOC) calculates the number of immediate subclasses subordinated to a class in the class hierarchy.
7	DIT	Inheritance	Depth Inheritance Tree (DIT) calculates the longest path from the class (in the class diagram) to the root of the inheritance tree.
8	CLD	Inheritance	Class Leaf Depth (CLD) calculates 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 (import)	The number of times the class is externally used as attribute type.
12	IC_Attr	coupling (export)	The number of attributes in the class have another class or interface as their type.
13	EC_Par	Coupling (import)	The number of times the class is externally used as parameter type.
14	IC_Par	coupling (export)	The number of parameters in the class have another class or interface as their type.

In part A, we aimed to discover the respondent's personal characteristics and experience with class diagrams. Meanwhile, Part B aimed to discover what object-oriented design metrics that the respondents find influential in considering classes that could be included in class diagrams. In part C, we aimed to discover what classes the respondents leave out when looking at a diagram and what class diagram(s) the respondents prefer when looking at different types of class diagram designs. This is an online questionnaire and is hosted by LimeSurvey [5] and a printable version is available at [130].

Table 6.2: *Answers Multiple Choices Questions*

Choices	Answers
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

Part A: Background of the Respondents

Part A consisted of 4 questions. Question 1 asked about the current status of the respondents. Question 2 intended to collect information about the respondent's location (optional question). We asked how many years of experience the respondent has with class diagrams in question 3. The last question asked the respondents to rate their skills in creating, modifying and understanding class diagrams.

Part B: Class Diagram Indicators for Class Inclusion /Exclusion

This part consisted of 14 questions. The first 13 questions asked about the influence of class diagram elements (based on object-oriented design metrics) to distinguish classes that should be included or excluded. In detail, we asked opinions of software developers on to whether they believe a particular metrics should be used for deciding whether a class should be included or excluded. In each question, we briefly explained about the metrics that was used in the question and five answers were offered. The choices of answers are shown in Table 6.2.

The last question of part B (i.e. question 14) is to discover the reason of the respondents for including and excluding a class in a class diagram. This question aimed to collect the information complementary to object-oriented design metrics about the reason of the respondents for including and excluding a class in a class diagram.

Part C: Practical Simplification Problems

Part C contained 6 questions. In this part, 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:** We used the forward design of an ATM simulation system [28] that only contains class names and class relationships. 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 which was taken from [55] contains 24 classes. The RE-CD of this system was used in this questionnaire.

Table 6.3: *Description of the Class Diagrams Used in the Questions*

Question	System	Source of Diagram	Level of Detail (LoD)
C1	ATM Machine	Forward Design	Low
C2	Library System	Reverse Engineered	High
C3	Pacman Game	Forward Design	High
C5	Pacman Game	Reverse Engineered	High

Table 6.4: *Choices of Answers for Question 4*

Choices	Descriptions
A	I prefer class diagram A (ATM System)
B	I prefer class diagram B (Library system)
C	I prefer class diagram C (Forward design Pacman)
D	I prefer them all
E	I do not prefer them
F	It does not matter which one

3. **Pacman Game:** Pacman's Perilous Predicament is a turn-based implementation of the classic Pacman game. To accommodate its turn-based nature, the game play mechanics will be changed into more of a puzzle game. This project can be found at [44]. In this questionnaire, we used the diagram of the second phase (Milestone 2). We used two types of diagrams from this system, namely the forward design and the RE-CD. The forward design consists of 17 classes while the RE-CD contains 15 classes. The forward design was detailed and hence, similar to the source code.

We also tried to simulate the various flavours of class diagrams from the software industry by providing different Levels of Detail (LoD) of class diagrams and the sources of class diagrams. Different flavours of class diagrams allowed us to differentiate the indicators of class exclusion. The information about the class diagrams used in part C is shown in Table 6.3. 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 part C), the respondents were required to choose between an ATM system, a Library system and the forward design of a Pacman system. The respondents were also required to provide the reason they chose the answer. In the second question (question 6 in part C), the respondents were required to choose between the forward design and the RE-CD of Pacman. The respondents were also required to give the reason they chose the answer. Table 6.4 and 6.5 show the answer options for the multi-choice questions.

Table 6.5: *Choices of Answers for Question 6*

Choices	Descriptions
A	I prefer class diagram C (Forward design Pacman)
B	I prefer class diagram D (Reverse engineered design Pacman)
C	I prefer them Both
D	I do not prefer them
E	It does not matter which one

Table 6.6: *Total of Reponses*

Responses	Amount
Complete Responses	25
Incomplete Responses	73
Total Responses	98

6.4.2 Experiment Description

The experiment was conducted online (hosted by Limeservice [5]). The questionnaire was published online from 15th of May until the 3rd of August 2012.

We first invited 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 LinkedIn. We also promoted this questionnaire to multiple online software developer forums.

The respondents were provided the facility to save the answers and the respondents could continue for a later time. The total respondents that entered this questionnaire were 98 (see Table 6.6). However, only 25 respondents completed this questionnaire. Most of the incomplete responses stopped after the questions in Part A.

6.5 Results and Findings

In this section, we present our results and findings from this survey. This section is divided into three subsections: Background of the Respondents, Indicator for Class Inclusion and Practical Simplification Problems.

6.5.1 Background of the Respondents (Part A)

This subsection presents the results of part A of the questionnaire in which we asked after the respondent's background information.

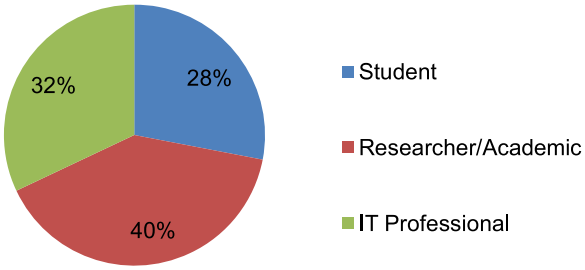


Figure 6.1: *Role of the Respondents*

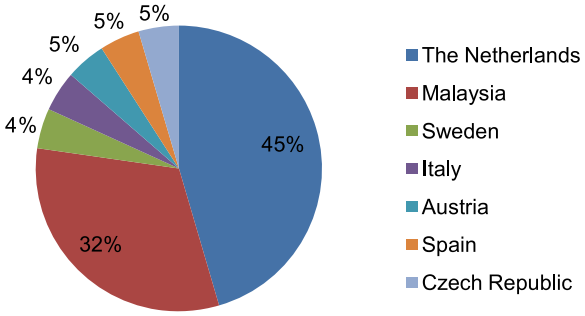


Figure 6.2: *Location of the Respondents*

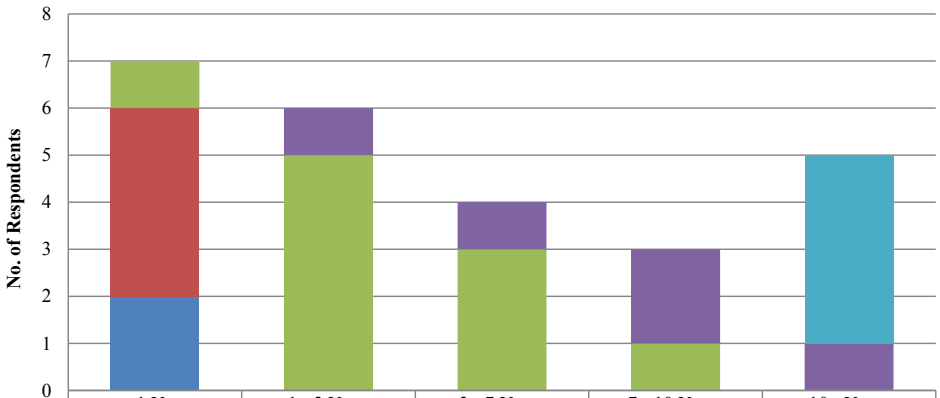
Roles and Locations

For the respondents' role, 40% of the respondents mentioned that their current status is Researcher/Academic while 32% of the respondents are IT Professionals. 28% of the respondents answered Student. None of the respondents answered "Other". Figure 6.1 shows the results of all the respondents. This result shows that the distribution of the respondent's status is quite even.

For the respondents' location, 45% of the respondents were from the Netherlands and 32% of the respondents were from Malaysia. The detail of respondents' location is shown in Figure 6.2.

Skills and Experience with Class Diagrams

For the years of experience in using class diagram, 28% of the respondents stated that their experience with class diagrams 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 had more than 10 years of experience with class diagrams.



	< 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 6.3: *Class Diagram Skill and Years of Experience*

In Question A4, we asked the respondent to rate his/her skills in creating, modifying and understanding class diagrams. 40% of the respondents answered Average, while 20% answered "Good". 16% of the respondents answered Excellent. 16% of the respondents rated their skill are Low and 8% of the respondents have Poor skill of class diagrams. This indicate that 76% of the respondents rated their skill of average or above. The complete results of the combination of these two questions are shown in Figure 6.3.

6.5.2 Indicator for Class Inclusion

In this subsection, we present the result of part B. This part consisted of 14 questions. Each of these questions asked whether some metric-value could be used as an indicator of the importance of a class. For example,

" **B3**: *Do you think that a high number of Public operations (NumPubOps) is an indicator that a class should be included in a class diagram? "*

For questions B1 to B13, the respondents were provided 5 answer options (see Table 6.2) to be chosen as their answer. We analyzed these 13 questions by using a score-system. The score-system is shown in Table 6.7.

The metrics are grouped into three categories which are: Size, Coupling and Inheritance. The details of these metrics are explained in Table 6.1. Figure 6.4 shows the results of the Size category. The average score for the metrics in category of size is 18.3.

Table 6.7: *Score-System Metrics - Question B1-B13*

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

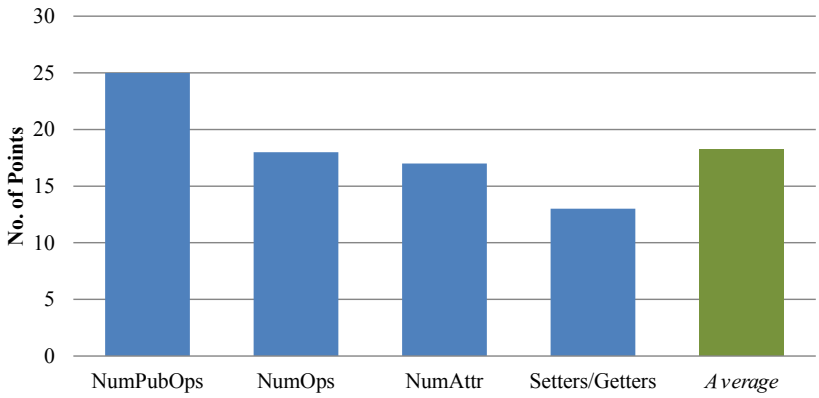


Figure 6.4: *Score Size Category (Question B1-B4)*

From these results, we found that operations are very important in class diagrams. In particular, public operations. This finding aligns with our findings in Chapter 5 where the respondents did not like to see private and protected operations. In other words, they find public operations better indicators for inclusion in a class diagram. As for setters/getters, these have the lowest score 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. NumAttr and NumOps also have an average amount of points. We can say that these metrics are normally needed in a class diagram, but public operations are more preferred.

Figure 6.5 shows the results of the Coupling category. On average, the score for the metrics in coupling category is 14.2. The results illustrate that Dep_Out and Dep_In score 17 and 16 points, respectively. In Chapter 5, most of the respondents find the class relationship is an important criteria of classes that should be included in class diagrams. 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)

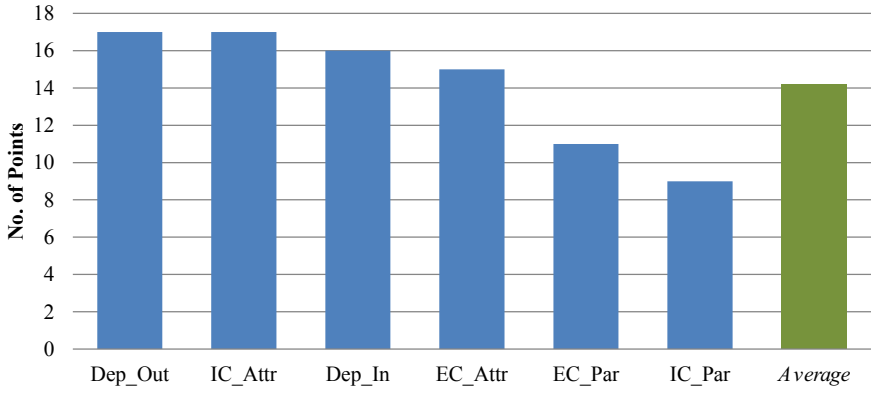


Figure 6.5: *Score Coupling Category (Question B5-B10)*

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.

The Inheritance category consists of three metrics: NOC, DIT, and CLD. The average score for the metrics of this category is 10.7. From the results (Figure 6.6), NOC has the highest score in this category (20 points). DIT and CLD have 7 points and 5 points, respectively. These results suggest that the respondents may only be interested in a part of an inheritance hierarchy. If a class has a high NOC, the class is important since it has many immediate children and is also higher in the inheritance hierarchy. However, 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 a low score because normally if a class has a high number of CLD then the class presents a very high-level of abstraction that is typically used to group the classes under this class.

Question B14 asked the reasons for including or excluding a class from the class diagram. The responses to open-ended question B14 were analyzed by grouping the answers into categories. An answer to this open-ended question could contain multiple keywords. The respondents stated that they wanted to include a class "when it is important" but they did not say when a class is important. It is a weakness of the survey-method that we could not ask for further questioning into more explicit factors when this answer was given.

Figure 6.7 shows the results of the question based on the keywords. It shows that there are three keywords that are related to the answers the most. These are Important/Relevant Class (29.6%), Domain Related (25.9%), and Coupling (18.5%). The keyword "Important/Relevant" is a broad term, but that is what the respondent answered. Hence, this answer is really obvious, but we cannot use it as a recipe to

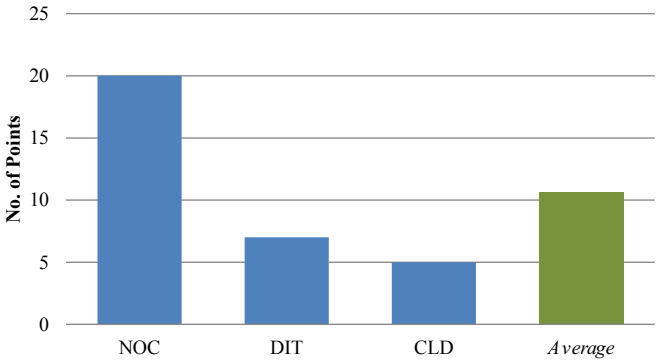


Figure 6.6: *Score Inheritance Category (Question B11-B13)*

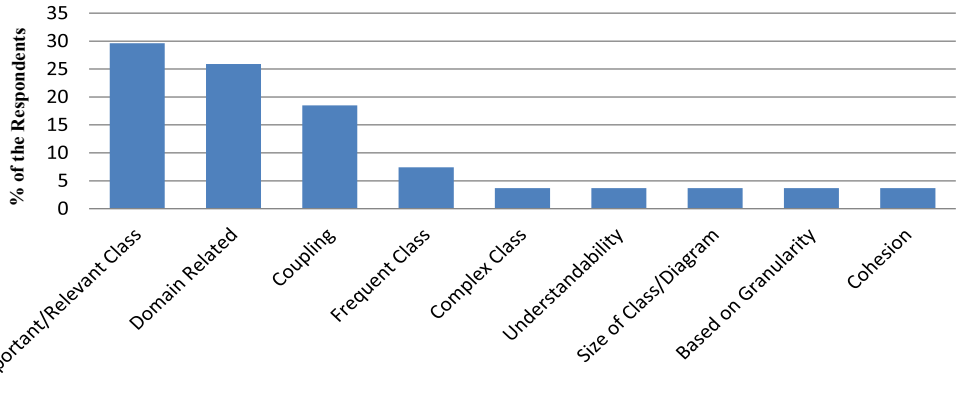


Figure 6.7: *Keywords to Include a Class in a Class Diagram*

decide which class is important for them. Coupling on the other hand is a factor that was expected. We stated in the survey in Chapter 5 that relationships are very important to understand a class diagram. Here, we found the same result. 18.5% of the respondents said that if a class has many relations, then that class should be included.

"Domain Related" are classes related to the concept or domain. Without these classes, it is hard for a software maintainer to understand a system. Five respondents mentioned the reasons for exclusion. One of these reasons is when a class is too small or 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 did not state when an attribute or an operation is important. One respondent stated that he did not need any children classes. In 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 to show the needs of the class names and class relationships 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.

6.5.3 Practical Simplification Problems (Part C)

In this part, we tried to elicit characteristics about classes that should not be included in a class diagram. This information is gathered by asking the respondents to select classes that should be excluded from the class diagram of three actual system designs.

Coupling

In question C1, a class diagram of an ATM System (as shown in Figure 6.8) was presented without attributes and operations. Through this question, we aimed to elicit information about the influence of coupling category and class names. The overall results of this question are shown in Figure 6.9. The results show that 48% of the respondents chose to exclude the class Money and 36% of the respondents chose to exclude the OperatorPanel and Status class from the class diagram. We observed that these 3 classes have a relatively low coupling (≤ 2). Next, 32% of the respondents excluded the classes Deposit, EnvelopeAcceptor, ReceiptPrinter, Transfer and Withdrawal. The coupling of those classes is equal to 2. This means that the exclusion of 8 out of 24 classes of this class diagram could be explained based on their coupling. The classes that were important in this class diagram are Transaction and ATM. Only 4% of the respondents chose these classes as should not be included. Both classes have a high amount of coupling. This indicates that the amount of coupling plays a major role in selecting the classes that should or should not be included in a class diagram.

Meaningful Class Names

A RE-CD from a Library System (as shown in Figure 6.10) was used for question C2 ("*Figure 10*" in this questionnaire). All elements in a class diagram were presented (in HLoD) and we expected to discover the factors that are influential in selecting the classes that could be excluded. The results of the survey are shown in Figure 6.11.

From question C2, 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 referred to functionality that is used to display information. Thus, these classes are not considered important because they are only

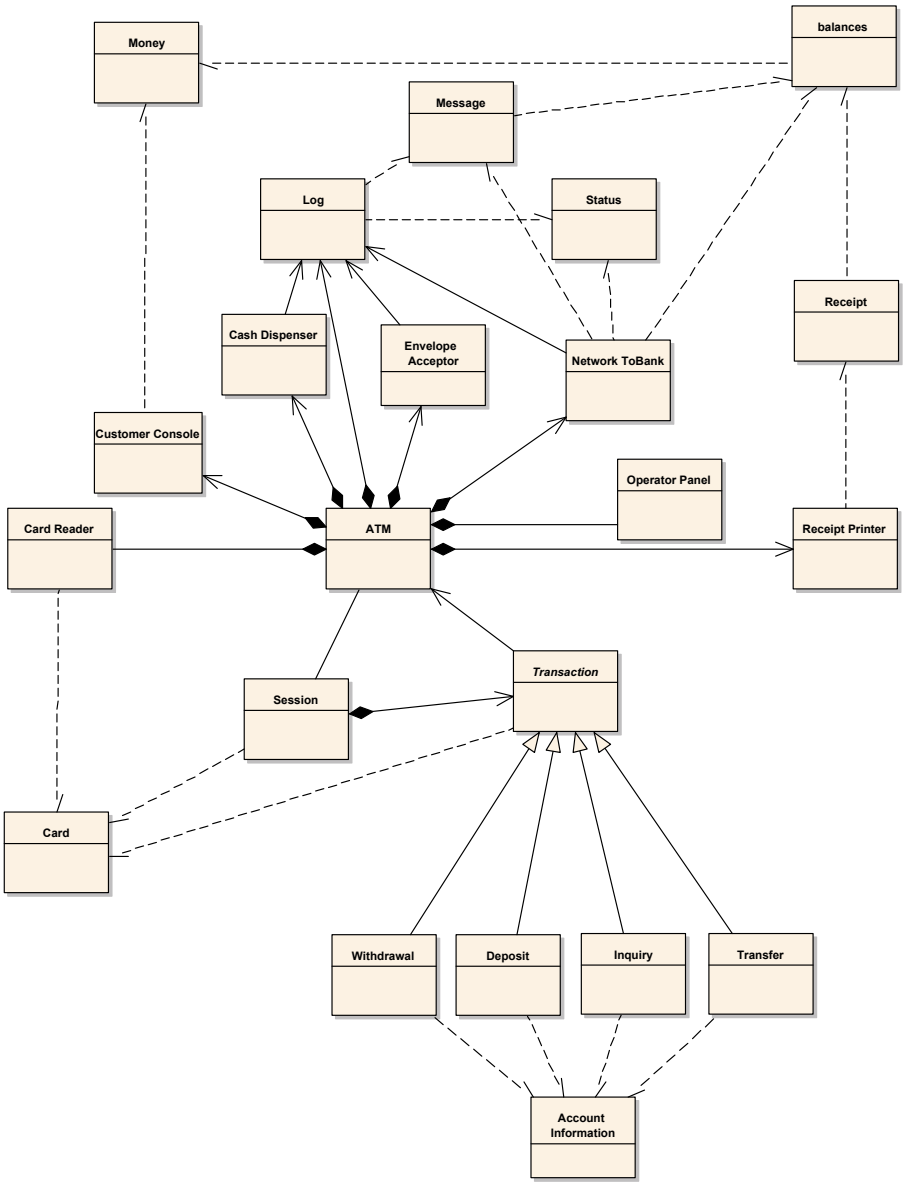


Figure 6.8: *Class Diagram A (ATM System)*

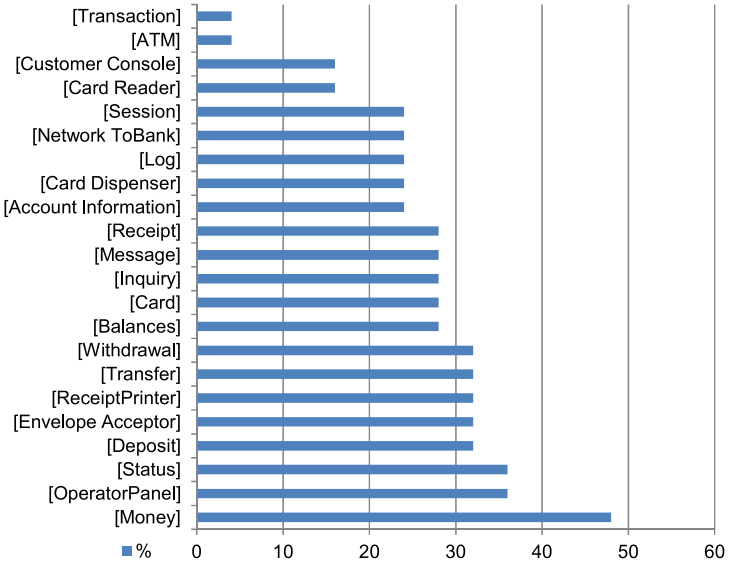


Figure 6.9: *Respondents Selection of Classes that Should not be Included in an ATM System*

used to display messages - which is not core functionality of this application. On the other hand, the 5 classes that not many respondents chose to exclude from the 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 are core concepts of the domain i.e. Library System.

Enumeration and Interface Classes

In question C3, the respondents were asked to select the classes that could be left out of a forward designed Pacman Game class diagram. Most of the classes in this diagram have relationships and meaningful class names. The complete result of this question is presented in Figure 6.12. The results indicate that 64% of the respondents chose to exclude class Direction from the class diagram. This class is an Enumeration class with coupling equals 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 to exclude the Iterable class. Both classes are interface classes that might indicate that those classes are not important or at least not related to the domain of the application. These results illustrate that the enumeration and interface types of classes are candidates for suppression in a simplifying this class diagram.

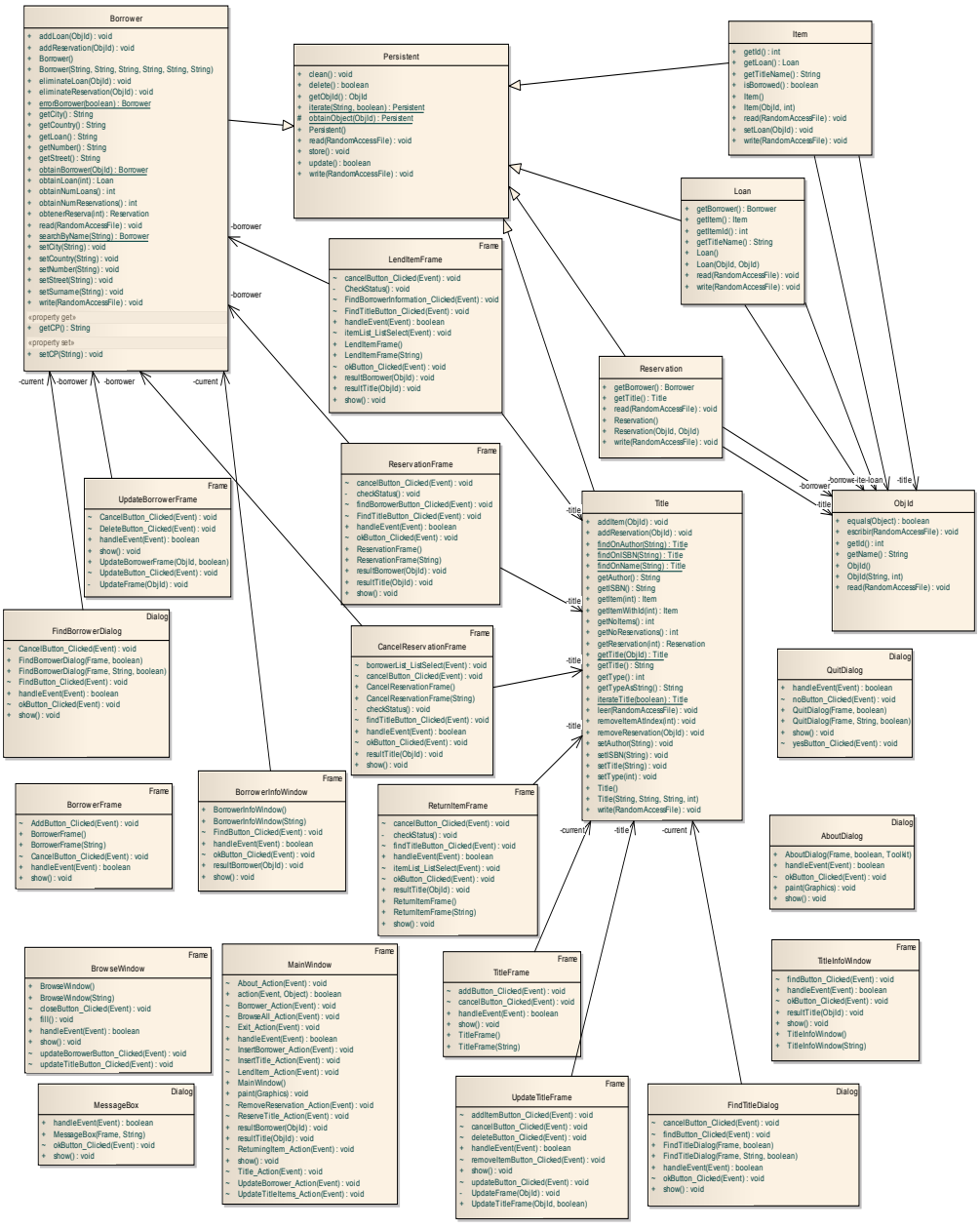


Figure 6.10: *Class Diagram B (Library System)*

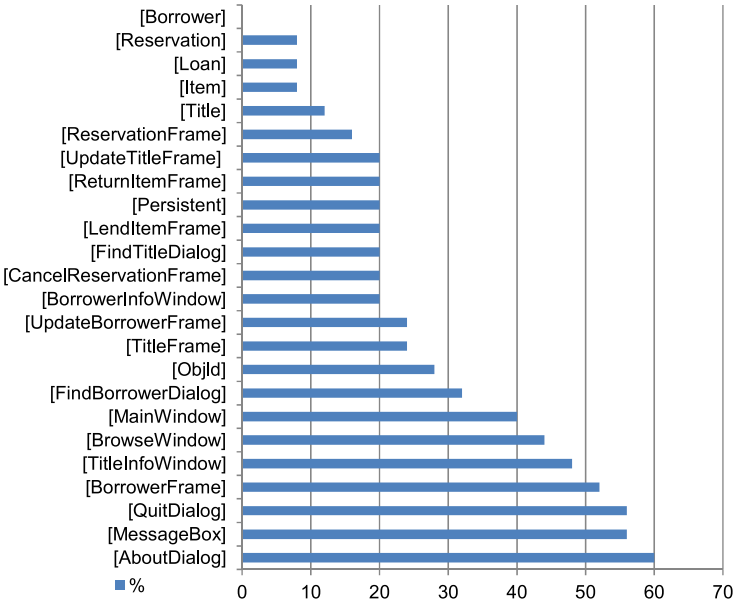


Figure 6.11: *Respondents Selection of Classes that should not be Included in a Library System*

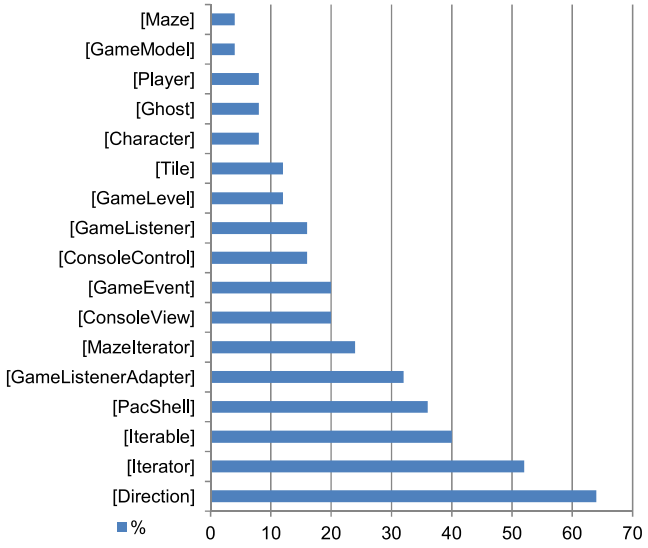


Figure 6.12: *Respondents Selection of Classes that should not be Included in a Pacman Game (Forward Design)*

Table 6.8: *The Preferences between Class Diagram A (C1), B (C2) and C (C3)*

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 6.8)	5	20	0	2	3	0	0	2	1	2
I prefer class diagram B (Figure 6.10)	2	8	0	1	1	0	0	2	0	0
I prefer class diagram C (Figure 6.13)	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**

Level of Detail

Referring to the class diagram A (question C1), class diagram B (question C2), and class diagram C (question C3), the respondents have been asked which flavour of the class diagram is preferred to be used. The detail information of LoD for the class diagrams is explained in Table 6.3.

The results in Table 6.8 show that almost half of the respondents (48%) preferred working with class diagram C (see Figure 6.13). This diagram is a HLoD forward design class diagram. They mentioned that the class diagram is clear, the necessary information is provided (e.g. attributes and operations) and most of the presented classes are important. This diagram was preferred 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 diagrams 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. 8% of the respondents preferred class diagram B. Another 8% did not prefer any of the

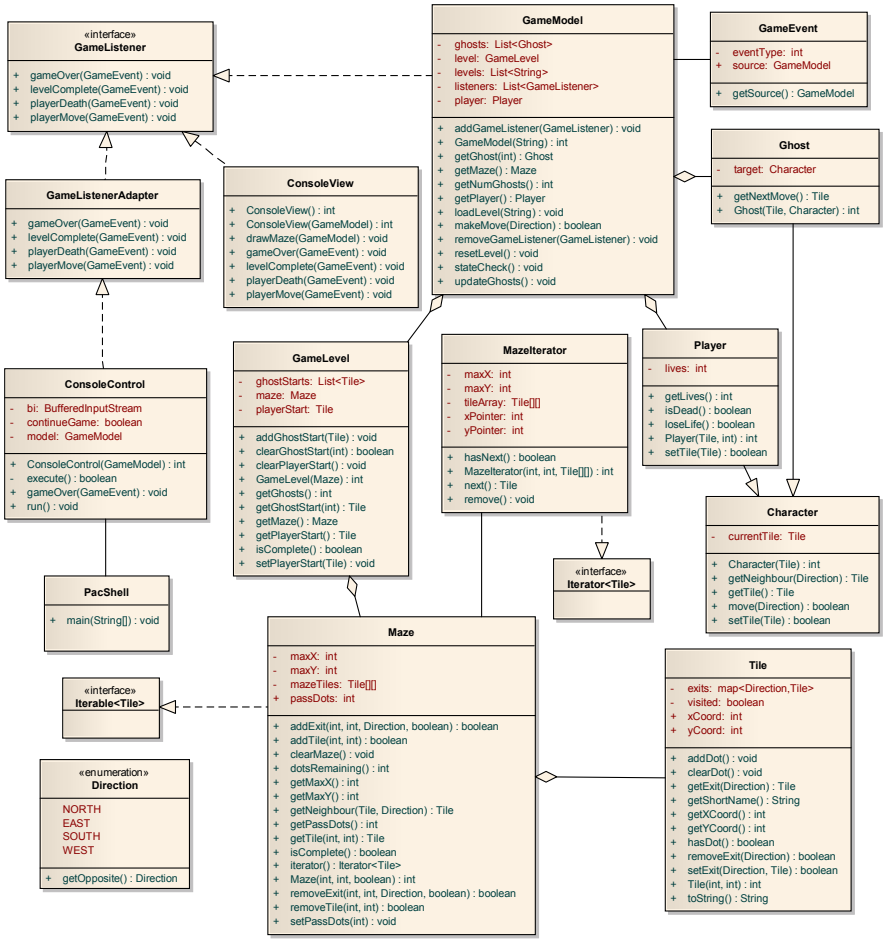


Figure 6.13: *Pacman Game Forward Design (Class Diagram C)*

presented class diagrams. As reason for this, they stated that there is "no story" in the class diagrams and that the class diagrams only show the solution, not the foundation of the domain.

Reverse Engineered Class Diagram which Conformed to Forward Design

In question C5, the class diagram used was slightly different from the class diagram presented in question C3 because this class diagram was constructed by using a reverse engineering technique (see Figure 6.14). In this question, we tried to discover if there was any difference in selecting the classes that should not be included in a class diagram in a Re-CD that is close or almost similar to the forward design class. The result shows that the class Direction and PacShell were selected by 72% of the respondents to be left

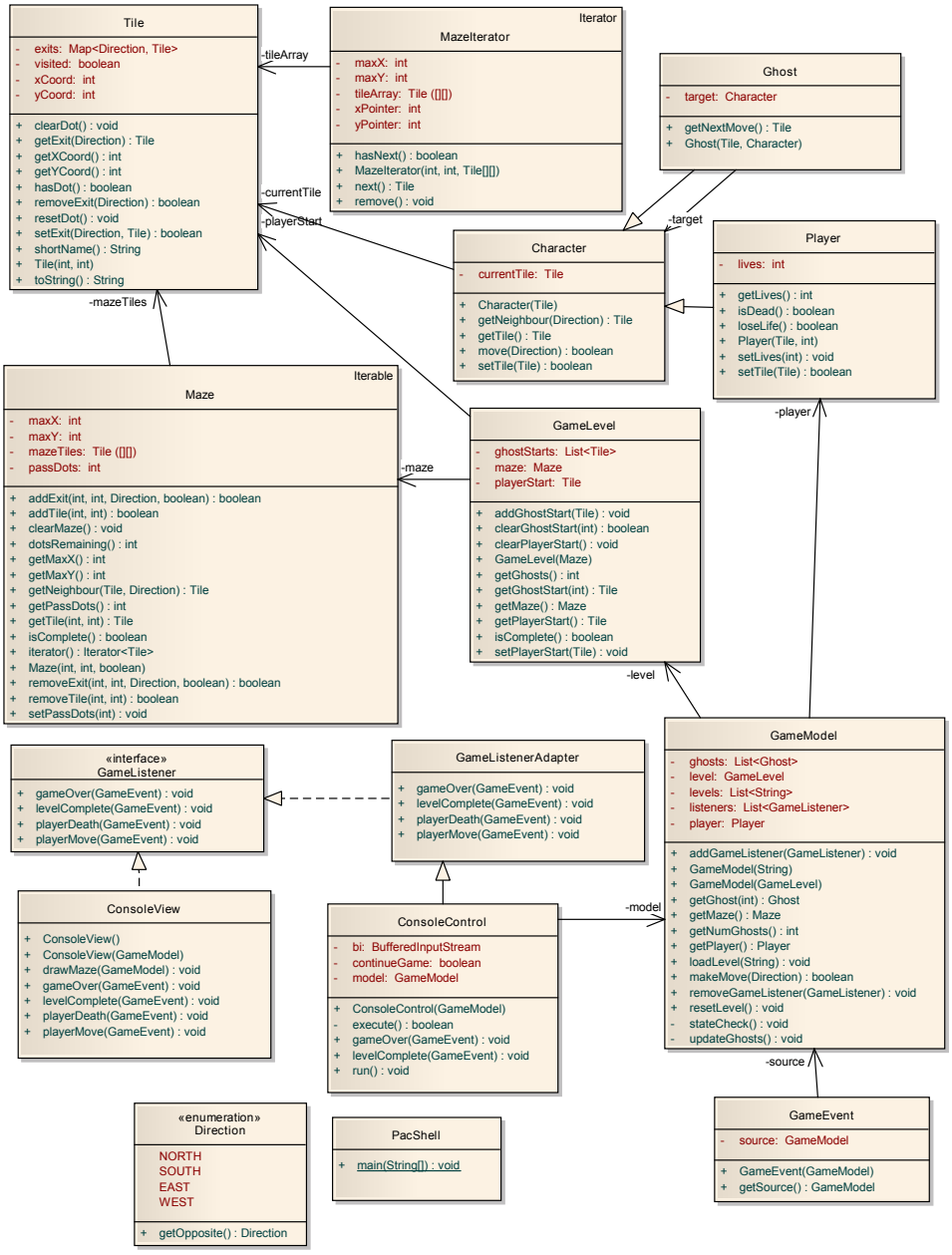


Figure 6.14: *Reverse Engineered Pacman Game (Class Diagram D)*

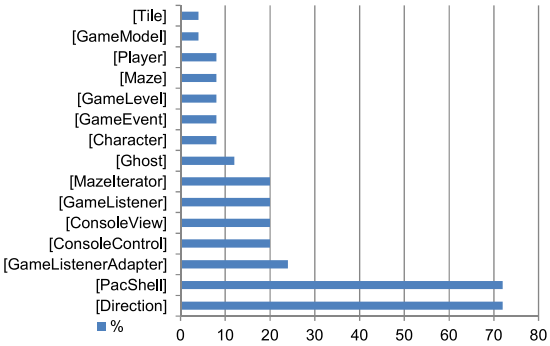


Figure 6.15: *Respondents Selection of Classes that should not be Included in a Pacman Game (Reverse Engineered Design)*

out from the class diagram. Compared to question C3, the Iterator and Iterable classes were differently presented in this reverse engineered diagram. The complete result of question C5 is shown in Figure 6.15.

Reverse Engineered vs. Forward Engineer Class Diagram

Question C6 aimed to discover which type of class diagrams was preferred by the respondents i.e. RE-CD or forward designed class diagrams. The RE-CD used in this question was different from the RE-CD in question C2 because this RE-CD was derived from a system that was implemented (or coded) closely with the forward design. The results in Table 6.9 show that most of the respondents (mainly researcher) preferred the class diagram D (the RE-CD from question C5 as illustrated in Figure 6.14). 40% of the respondents chose this diagram because it is more detailed, clear, interface classes and it is easier to understand. 20% of the respondents did not choose any of the two class diagrams because they did not have a preference. 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 the class diagram C (from question C3). There is no pattern of selection present in this result in terms of the respondents' role and skill.

If we compare the results of this question and the results of Section 6.5.3, we found that RE-CD is chosen if the source code was closely following the forward design, because then the difference between forward design and its RE-CD is small.

6.6 Discussion

In this section, we discuss the results and findings presented earlier in this chapter. This section is divided into 5 parts which are: Respondents' Background, Software

Table 6.9: *The Preference between Class Diagram C and D*

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 6.13)	10	40	1	7	2	1	1	5	2	1
I prefer class diagram C (Figure 6.14)	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**

Design Metrics, Class Names and Coupling, Class Diagrams Preferences, and Threats to Validity.

6.6.1 Respondents' Background

In Part A, the respondents' profiles to this questionnaire were quite evenly distributed. The location of the respondents showed that most of the respondents are from The Netherlands and Malaysia. This is due to the professional and personal network of the author.

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 with UML 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 had less than one year of experience, we can still state that all the respondents have knowledge about class diagrams.

6.6.2 Software Design Metrics

We asked which metrics could be used as indicators for inclusion or exclusion of classes from class diagrams. We classified metrics in 3 categories: Size, Coupling and Inheritance. The overall ranking of the score is shown in Table 6.10.

Table 6.10: *Overall Score for Software Design Metrics*

No.	Design 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

In the Size category, we found that the higher number of public operations, is the more people prefer this class. There is a clear preference for public operations over operations in general because public operations are considered to more often stand for important functionality.

In the Coupling category, we found that the classes that have many incoming and outgoing dependencies are preferred for inclusion. We found that IC_Attr and EC_Attr have higher scores than EC_Par and IC_Par. The reason might be that a class that is declared as an attribute is more important because it can be used across many operations in the class.

In the Inheritance category, we discovered that for a class that has a high NOC, the class should be included in a class diagram. This parent class is helpful to show the abstraction of a group of classes. For DIT, a 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 value for this metric this that this class is very abstract, meaning that this class alone will not be enough to understand the whole hierarchy.

As for the complete results, we found that NumPubOps has the highest points across all metrics. Also, all the metrics received a positive score, even though some only slightly, while a negative score is also possible. Hence, each of these can be considered in our subsequent studies on how to best use these metrics in selecting classes for in/exclusion.

6.6.3 Class Names and Coupling

In Part C, we showed that coupling is a highly influential factor when we are trying to exclude classes from a class diagram. Another influencing factor is the class name. From our observation, the class names are an influencing factor since it may indicate the class role and responsibility. Through class names, the respondents make assumptions of the class functionality (role, responsibility, service) and as well as the flow of the system. Many respondents excluded Graphical User Interface (GUI) related classes in the Library system because of the class role and responsibility (class name based), and coupling.

Aside from these two big influencing factors, many respondents excluded types 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 generic and not key for the domain of an application. Overall, we suggest that the role and responsibility of the classes (based on class names) could be an influencing factor in deciding class inclusion/exclusion.

6.6.4 Class Diagram Preferences

In question C4, we found that most of the respondents preferred to use HLoD of the forward design. The reasons the respondents gave was that this class diagram is clear and the necessary information is provided in this class diagram. Meanwhile, in question C6, most of the respondents had chosen the RE-CD (HLoD). The reason might be that the RE-CD that was provided has few differences from the forward design. The respondent stated that they preferred this diagram because they find it more detailed and it is easier to understand. Some of the respondents also mentioned that the interface classes are removed, which make it a better class diagram.

From our observation, the RE-CD 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.

6.6.5 Threats to Validity

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

6.7 Conclusion

In this survey, we revealed the metrics that indicate the classes that could be left out. We also found the flavour of class diagrams that developers prefer to work with. From the results, we found 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 found that the class names and coupling are influencing factors when selecting a class to be excluded from a class diagram.

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

6.8 Future Work

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

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