CHALMERS
UNIVERSITY OF TECHNOLOGY

Condensation of class diagrams using machine learning

Master’s thesis in Software Engineering

FILIP BRYNFORS

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Condensation of class diagrams using machine learning

FILIP BRYNFORS

Examiner: ERIC KNAUSS
Supervisor: MICHEL CHAUDRON, TRUONG HO QUANG

Masters Thesis 2016:NN
Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2016

Abstract

Although class diagrams can be a useful asset within software development, the size of reverse engineered diagrams may quickly become overwhelming in a large scale software environment. One way of keeping diagrams small is to condense them by removing the least important classes while keeping the remainder. This can be done through the use of machine learning where metrics of a class are used determine how important the class is. As not very many metrics have been used in this setting, the purpose of this thesis is to identify what metrics that are considered to be relevant for different stakeholders and then to find how well the identified metrics work within the machine learner. Furthermore, the thesis also aims at evaluating if the method as a whole is useful within an industrial setting. The work was started by interviews in collaboration with Ericsson where their opinion regarding the importance of classes was gathered. Then followed the creation of a tool for condensing diagrams and an analysis of the performance of the machine learner. Additionally, condensed diagrams were compared to reverse engineered diagrams. Several new high performing metrics were found and extending the previous set of metrics improved the performance of the machine learner overall. Some flaws were identified with the condensed diagrams, however they still managed to perform better than reverse engineered diagrams with all classes remaining and thus, the method should be considered advantageous in an industrial setting.

Keywords: machine learning, condensing diagrams, metrics, class importance, reverse engineering

Acknowledgements

I appreciate the help of everyone who in any way, shape or form have contributed towards this thesis and I would like to express my gratitude towards all of these individuals. I am sincerely grateful for all the feedback, guidance and discussions with my supervisors Truong Ho Quang and Michel Chaudron. I am thankful for feedback and advice given by my examiner Eric Knauss. I would like to express my deepest gratitude towards the EPG department at Ericsson for giving me the opportunity to do this work, for all insights that I have been given and for all assistance that I have received throughout the work.

Contents

Chapter 1

Introduction

The documentation of the software architecture is not only important for creating the software, it is also very favourable to use as a means of communication and as a tool for learning throughout the lifetime of the software [1]. Thus, maintaining the documentation as the software evolves is certainly beneficial. Nevertheless, this does not always happen in practise. Additionally, manually creating new documentation of the architecture of the software repeatedly as it evolves is a time consuming task.

One way to deal with this issue is to automatically reverse engineer the source code into diagrams that represent the architecture of the software [2]. However, this method is not flawless. For any larger system, preserving all the elements would result in diagrams of massive size. Too much detail would be kept and the architecturally important elements would be indistinguishable and therefore, the diagrams would be rendered useless.

Thus, reverse engineering must be accompanied by a method to condense the diagrams in order to only preserve relevant elements and efficiently display the architecture of the software. While at the same time, removing too many elements renders the diagrams incomprehensible due to lack of information available for the reader.

One way of automatically condensing diagrams is through the use of machine learning algorithms [3]. Such algorithms use a set of software metrics as input. These metrics are called predictors as they are the key elements that the algorithm learns and uses in its predictions. In this case, the prediction should be of whether or not a class is important enough to stay in the diagram or not.

In the training phase, the algorithm is fed with manually created diagrams representing software systems without including all classes along with reverse engineered diagrams that do contain all the classes of the same systems [4]. By doing this, the machine learner is able to learn how to select the important classes from the reverse engineered diagram, based on the values of the predictors.

In the application phase, the algorithm is fed with a new reverse engineered diagram and then uses the previously learnt knowledge in order to determine which classes from the diagram that are the architecturally most important and should be kept in the condensed diagram [4].

By letting the algorithm produce a value of importance for each class, rather than only a binary value indicating whether or not the class should be included, it is possible to sort the classes by importance and make condensed diagrams at various levels of abstraction. Including very few of the most important classes would produce a high level diagram while including many of the most important classes produces a diagram of lower abstraction level.

1.1 Problem

At the moment, the set of software metrics that are used as predictors is very limited [3, 4, 5]. These are explained further in section 3.1. Additionally, the condensed diagrams from condensation with one set of metrics may not be suitable for different kinds of stakeholders. E.g, a system architect might have different opinions regarding what classes that are seen as important compared to a developer.

Discovering what metrics that different stakeholders think are important could not only improve the quality of the condensed diagrams for different stakeholders, but creating a larger set of metrics to be used as predictors could increase the accuracy of the predictions for all stakeholders.

1.2 Purpose

The aim of this thesis is to, in collaboration with Ericsson, gain more knowledge regarding which software metrics that are perceived to be architecturally important within the design of class diagrams for various stakeholders within software development and use this knowledge to select metrics which advantageously can be used as predictors in a machine learning algorithm for condensing reverse engineered class diagrams. By including a larger number of predictors which are relevant to the architecture, the accuracy of the condensed diagrams from the machine learner could be improved.

Furthermore, this thesis presents a software prototype capable of condensing reverse engineered class diagrams using the selected software metrics. Not only should this prototype aid in validating the usefulness of the method of condensing diagrams through the use of machine learning algorithms, but it should also be able to be used within future research in this field.

1.3 Research questions

To address the problems described above, these three research questions are the focus within the thesis:

- RQ1: Which metrics of a software class are considered to be architecturally important for different stakeholders within the software development?
- RQ2: Which metrics of a software class can advantageously be used as predictors in a machine learning algorithm to determine the importance of the class?
- RQ3: Are class diagrams which are condensed through machine learning useful within industry?

RQ1 helps with gaining more knowledge within the area. Having information about what is important gives a good indication of where useful metrics can be found. However, while there may be a correlation between whether or not a metric is perceived to be important and if the metric can be used as a predictor, it is not certain that all the metrics that are considered to be important actually work well within the machine learner. RQ2, gives a direct indication of which metrics that actually are useful and should be used within the machine learner. RQ3 shows on a higher level whether or not the method of condensing diagrams as whole works in a real world setting.

Chapter 2

Background

In this chapter, both UML and machine learning are thoroughly explained as they serve as a base for the entire thesis. Additionally, the company, Ericsson, which the work is done in collaboration with is introduced.

2.1 UML

Unified Modeling Language (UML) is a language for defining and visualising software models [6]. It was created in 1995 and has been an ISO standard since 2005 [7]. While UML defines a lot of different building blocks, UML diagrams can be divided into two main groups: diagrams describing the structure of the software and diagrams describing the behaviour that the system has [6].

2.1.1 Structural diagrams

One of the diagram types that statically show the architectural structure of the software system is the class diagram. A class diagram shows the classes of the system along with the relationships between the classes [6]. A very simple class diagram with four classes can be seen in figure 2.1.

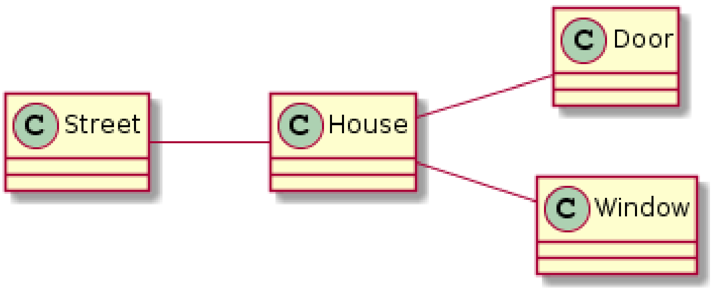

Figure 2.1: A very simple UML diagram generated in PlantUML

A class can be further extended to include attributes and operations. The Door class which can be seen in detail in figure 2.2 has the attributes height and colour as well the operations open and close.

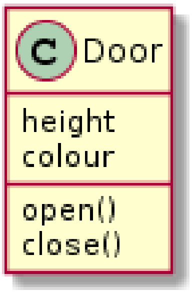

Figure 2.2: A class with attributes and operations generated in PlantUML

Furthermore, several types of relationships exist. Simply using a line shows association while a filled diamond shows aggregation and an empty triangle shows generalisation. Additionally, relationships can be extended to include the multiplicity that are allowed between the related classes. These relationships can be seen in figure 2.3.

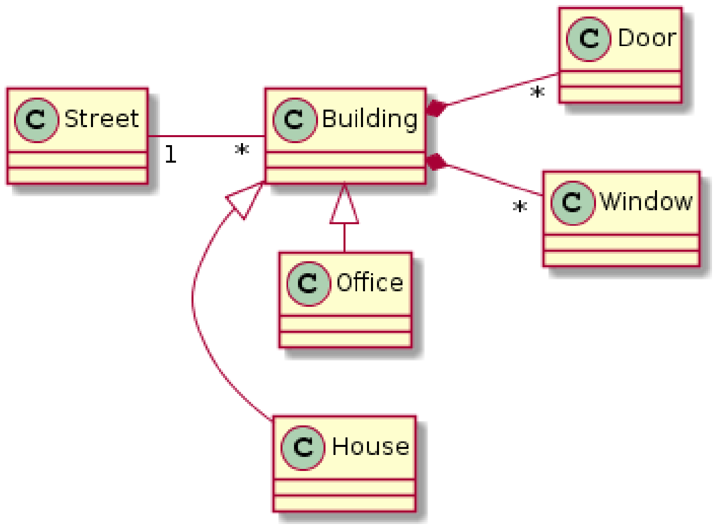

Figure 2.3: A class with attributes and operations generated in PlantUML

Other commonly used diagrams are component diagrams showing components and their relationships, object diagrams showing objects and their relationships and deployment diagrams that show the nodes that the system is deployed at and the relationships between those nodes.

2.1.2 Behavioural Diagrams

The behavioural diagrams how things are happening in the system. On of these diagrams is the Sequence diagram. These diagrams look at how messages travel between objects and in what time order [6]. A simple sequence diagram of how a student registers for a course can be seen in figure 2.4.

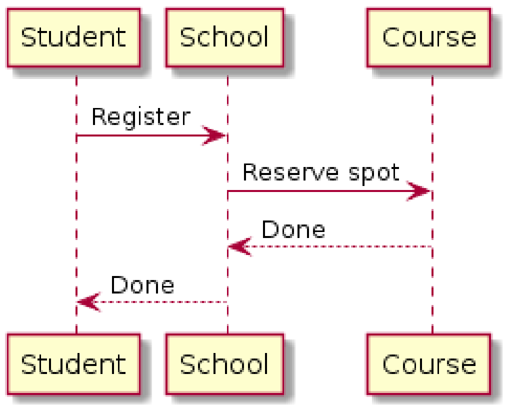

Figure 2.4: Three classes with calls between them generated in PlantUML

Further diagrams that show the behaviour are use case diagrams which show how actors interact with the system and activity Diagrams that show how the flow goes between different activities in a system. Also, statechart diagrams show the different states a system is in, as well as the possible transitions between states and events that can trigger the transitions. However, this thesis only looks at the architectural structure and focuses on class diagrams.

2.2 Machine Learning

Machine learning denotes a very broad field of research. However, it all regards intelligent software that is able to learn information and draw new conclusions based on previously inputted data. There are many ways to achieve such software, and many concepts that are involved.

The learning within the software can be divided into two different groups: online learning and offline learning [8]. In offline learning, there is an initial learning phase where a set of data is used to train the software. After that, the software is able to directly draw conclusions in new situations. In online learning however, the learning phase never ends. Whenever the software is used to draw conclusions in new situations, the information regarding the new stituation is also utilised within the software to further improve the results of future use.

Furthermore, machine learning can also be separated into supervised, unsupervised and reinforcement learning [9]. In supervised learning, the software is given a set of data to learn, but also the correct output. Then the software is capable of mapping given input with the wanted output. However, in unsupervised learning, the correct output is not given. Instead, the software may be able to find new information and draw conclusions which have previously been unknown.

Reinforcement learning works very differently. During the training, the software is not given a set of data, but instead it is given a value of how well it performs in a certain situation. Additionally, it is up to the software itself to discover how it should be tweaked to produce the best result. One could say that this is using the method of trial and error.

Yet another way of approaching machine learning is to divide it into the types of output it produces [10]. Different types of output could be for example a linear model of the data, a rule for classification of new data or a clustering of the inputted data. In this thesis, the machine learning used is supervised with offline learning to solve a classification task.

2.3 Ericsson

One of the largest communication technology companies, Ericsson [11], has undergone a change of software development platform within the EPG department. From previously using a Rhapsody based platform, they have moved towards using an Eclipse based platform.

Rhapsody has been part of the Model Driven Engineering process and was responsible for creating both architecture models and software implementation [12]. Thus, Rhapsody helped architects to keep track on changes in the software architecture models (top-down approach). However, what is lacking is visualisation from actual implementation (bottom-up approach) that is suitable for different stakeholders. A tool that can complement Rhapsody's current views would aid in the architectural decision making process.

Chapter 3

Related works

3.1 Condensing using machine learning

Automatically creating condensed diagrams is not impossible. This is demonstrated by Osman et. al. who used different machine learning techniques in order to determine the importance of a class [4]. In their research, a set of software metrics were used as predictors in the machine learning algorithms. However, this set is very limited and extending it could improve the result of the algorithms, and thus also improve the condensation of the diagrams. Something that is consistent with the entire set of metrics used is that all metrics are design metrics of the software, such as the number of operations or the number of dependencies of a class. Development metrics such as the number of times as class has been changed or the number of contributors are not present at all.

Similarly, Osman et. al. used a machine learning technique in order to create condensed diagrams as well [5]. However, in addition to the limited set of software metrics as predictors, a set of predictors based on the class name was utilised. A better result was achieved using both sets of predictors than just using the initial set. This clearly shows that extending the set of predictors can improve the condensed diagrams. Thung et. al. had a similar approach where network metrics were looked into to extend the set of predictors [3]. Additionally, Steidl et. al. also used network metrics to find the most important classes for reverse engineering [13].

Another approach was taken by I. Sora and D. Todinca, who used fuzzy logic to find the most important classes in a system [14]. The metrics that were used in their work was however also limited. The number of methods, the weight of the incoming and outgoing dependencies and the PageRank value of a class was used as input for determining the importance.

Nevertheless, even though the possibility of creating these condensed diagrams exists, the availability of such tools is lacking. Furthermore, the number of tools with the capability of producing various views for different types of stakeholders is especially insufficient.

3.2 Other tools

A comparison of four reverse engineering tools that do not utilise machine learning for condensation was done by Belley and Gall [15]. This comparison included the tools Refine/C, Imagix 4D, Rigi, and Sniff+. In the comparison several groups of criteria were assessed: criteria for the parser of the source code, criteria for the representation of the generated information, criteria for the editing capabilities and the available views, and general criteria such as supported platforms and toolset extensibility.

In the analysis it was determined that all tools had strengths and weaknesses and none of the tools could be considered as the best. However, a problem with all tools is that the size of the system was so large that in the graphical views, only parts of the system could be shown. Additionally, complete views of the system could not be created because of technical reasons such as development in multiple languages or systems with a client-server pattern. Furthermore, traceability between views was lacking.

3.3 Class importance

A way of measuring the importance of a class was presented by Hammad et. al. [16]. Individual classes were considered more important the more design changes that had been made towards the class. These changes do not only include the class itself, but also changes of relationships between the class and other classes. Furthermore, the importance of groups of classes were also measured. During the research, it was found that not many classes actually are important for the design.

3.4 Layouting

Other ways other than condensing diagrams exist in order to increase the readability of diagrams. One way is to work with the layout of the elements of the diagram. Several criteria for judging the layout were presented by D. Sun and K. Wong [17]. Some of these criteria relate to the orientation of the diagrams and the symmetry that can be seen within them. Others, however are a bit more specific and include aspects such as joining inheritance arcs rather than having completely separate lines or using colours to distinguish different groups of entities from others. Also the length of edges and overlapping of elements are considered. While there may be a large variation in these criteria, all aim towards the same goal: increasing the usability and understandability of the diagrams.

Chapter 4

Methodology

There were several steps involved in the working process. These can be seen in figure 4.1. The first step was to gather information about what different stakeholders think makes a class important in a software project in order to find important software metrics. The second step involved creating a tool for condensing diagrams. The third step was to gather a ground truth for the importance of a set of classes. In the fourth step, statistics were applied to determine the usefulness of each of the available metrics. The last step was evaluation of the condensed diagrams created by the tool.

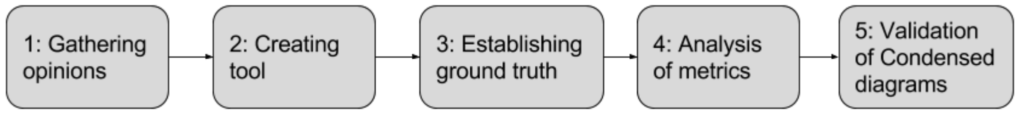
1: Gathering opinions
2: Creating tool
3: Establishing ground truth
4: Analysis of metrics
5: Validation of Condensed diagrams

Figure 4.1: The steps in of the work.

The work was carried out as a case study in collaboration with Ericsson. A case study is conducted in the real world and thus has a high degree of realism at the cost of control [18]. As part of this thesis regards validating the usefulness of the machine learning method for condensing diagrams, performing a case study is very appropriate.

4.1 Step 1: Gathering opinions

To address RQ1, the opinions regarding which metrics that are considered important were gathered through a case study in which the EPG department at Ericsson was studied. More specifically, five employees within the department were involved in the study.

These five employees were selected based on differences in roles and teams rather than trying to replicate the circumstances fully and just have one type of role in one team. Thus, it was possible to perform a cross-case analysis to see differences between the different roles of the employees [19]. Additionally, having multiple employees to be studied allowed for triangulation of data to improve the accuracy of the result [18]. The roles and teams in the study can be found in section 5.1.1.

4.1.1 Data collection

Data was mainly collected using qualitative methods. However, quantitative methods were also utilised to support any conclusions that were drawn.

Qualitative

As the intention was to extend the limited set of metrics while determining what people think makes a class important, exploratory means of gathering information were required. Semi-structured interviews is one tool of data collection which allows for usage of both open and closed questions and is thus suitable for exploring the opinions of what is important [20].

Interviews were held with each of the five selected employees at different occasions. Each interview was limited to a time period of 15 to 20 minutes. A tape recorder was utilised during the interviews in order to increase the correctness of the gathered data [20]. Furthermore, the recordings of the interviews were transcribed.

The questions asked during the interview followed an interview sheet that was made prior to the interviews. The interview sheet, which can be seen in appendix A, contained questions regarding the working process and working tasks in order to see the need of information the interviewees had from the software. Such questions included but was not limited to ”When there is a change of requirements so that you need to build new software or change existing software, how do you start working with the change?” and ”What information do you want to find when you read class diagrams”.

Additionally, the interview sheet also contained more direct questions aimed at looking at the importance of software classes. These questions included but were not limited to ”Would class with perhaps a high cyclomatic complexity or a large number of lines of code be more important for you?” and ”Does the fact that a class is changed by a lot of people make it more important to you?”.

Quantitative

Quantitative data was collected through a questionnaire. The questionnaire was sent to, and was answered by, the same employees that were interviewed. However, it was answered prior to the employees being interviewed.

The questions within the questionnaire regarded background information rather than being exploratory. Information that was gathered includes the current role and team, prior experience within software development and UML notations, and regularly used diagram types.

4.1.2 Data analysis

During the analysis of the data, the method of coding was used to group pieces of text. An editing approach was assumed [18], which means that a few priori codes were used. These codes were based on what was found and they were not defined prior to the analysis. The codes that were find are listed and explained further in section 5.1.

Furthermore, the codes were mapped with the metrics that were available. This was done by first grouping the metrics into groups where metrics measuring similar things were put into the same groups. Then the defined codes were linked to any metric group that was deemed to being able affect the code in any way.

4.2 Step 2: Creating tool

A tool that is capable of condensing diagrams through the use of a machine learning algorithm was created to aid in RQ2 and RQ3. This tool uses the k-nearest neighbour algorithm in order to learn in what situations a class is considered important and in what situations it is less important and to predict how important new classes are. This is done in two distinct phases; one learning phase and one prediction phase. The k-nearest neighbour algorithm with five neighbours was selected as this was the algorithm which performed the best when different algorithms were compared [4].

In the learning phase, several kinds of inputs are present and this can be seen in figure 4.2. First of all, a reverse engineered diagram is inputted. This diagram is reverse engineered from the source code using Enterprise Architect. From this diagram, the tool automatically extracts design metrics for each class. These design metrics include:

- NumAttr = The number of attributes a class has
- NumOps = The number of operations a class has
- NumPubOps = The number of publicly available operations a class has
- Setters = The number of Setter operations in a class
- Getters = The number of Getter operations in a class
- DIT = Depth of inheritance
- EC_Attr = The number of times the class is used as an attribute
- IC_Attr = The number of attributes in the class with another class as type
- EC_Par = The number of times the class is used as parameter
- IC_Par = The number of parameters in the class with another class as type

Additionally, for each class, the development metrics that were available are also inputted. These include:

- nloc = Number of lines of code for the last version during 2015.
- added = Sum of added lines of code during 2015.
- changed = Sum of changed lines of code during 2015.
- deleted = Sum of deleted lines of code during 2015.
- versions = Number of commits during 2015.
- prev_versions = Number of commits during 2014.
- prev_prev_versions = Number of commits during 2013.
- age = Number of years the class has existed.
- cyclomatic_complexity = McCabes cyclomatic complexity for the last version during 2015.
- token_count = Number of words in the class for the last version during 2015.
- parameter_count = Sum of all arguments to all functions in that class for the last version during 2015.
- contributors = Number of contributors during 2015.
- cumulative_contributors = Number of contributors since the class was created.
- defects = Number of defects during 2015.
- prev_defects = Number of defects during 2014.

Furthermore, a rating of how important a class is for understanding the package is inputted. This rating is an integer ranging from 1 to 10. This input serves as the ground truth and is explained further in section 4.3.

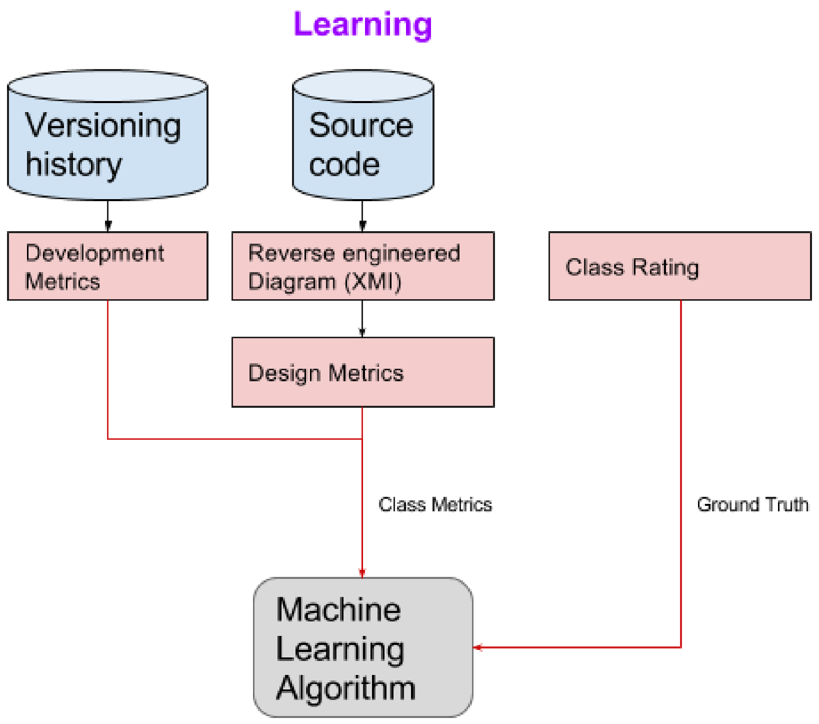

Figure 4.2: How the learning in the tool works.

The prediction phase is very similar. However, the difference is that in this phase, the class rating is not inputted at all. Instead, what the tool does is that it outputs a predicted rating for each of the classes in the reverse engineered diagram.

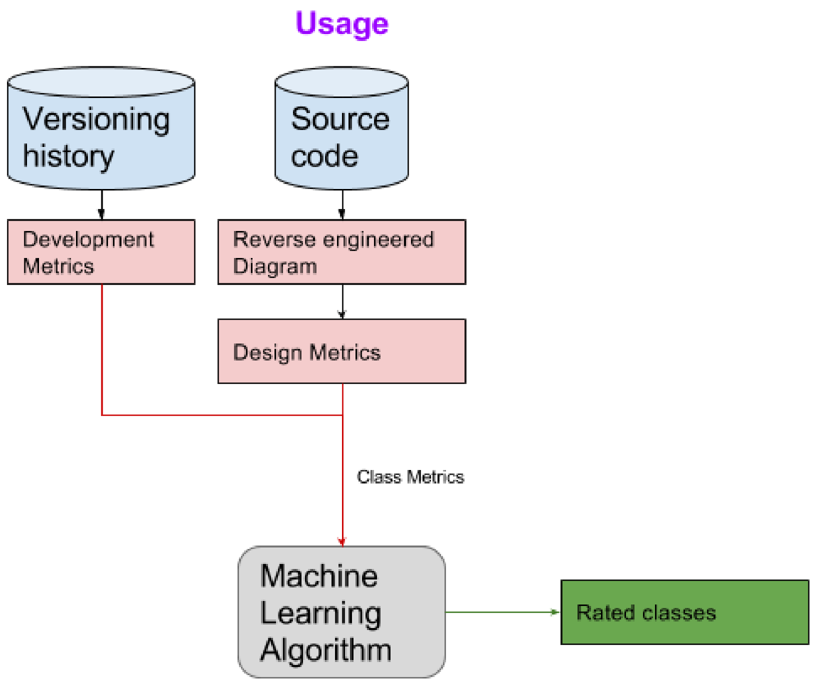

Figure 4.3: How the predictioning in the tool works.

All of the metrics that are inputted are normalised. The method of min-max normalisation was used to ensure that all values of all metrics lie within the range 0 to 100. Normalisation is cruicial for the tool as the distance measuring algorithm k-nearest neighbour is used [21]. This is easy to see when looking at for example the number of bugs in a class which is likely to be even lower than 10, while the number of lines of code in a class could be as large as a few thousand. The distance from differences in lines of code would then greatly outweigh the differences in number of bugs if the values are not normalised so that both metrics lie within the same possible range.

The tool is developed as an eclipse plugin. This is due to the fact that eclipse is a widely popular development platform which also is currently being used within Ericsson. The tool adds a view which contains the condensed diagram. This allows for the diagram to be accessible side by side to the source code while working.

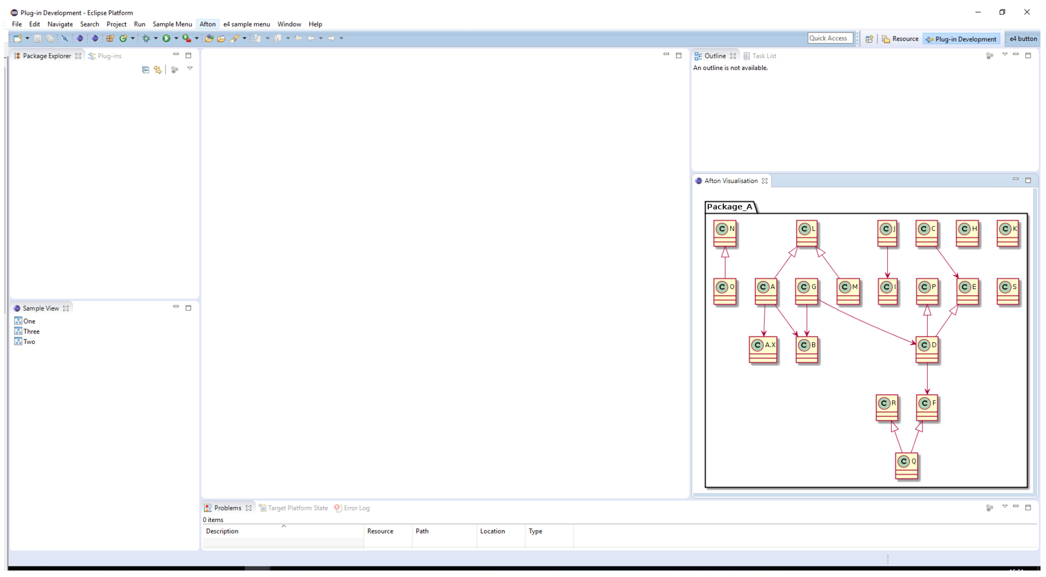

Figure 4.4: How the tool looks within the eclipse environment.

4.3 Step 3: Establishing ground truth

In order for the machine learner to be successful, the ground truth of which the tool learns from needs to be of high quality. A ground truth that does not match with reality would cause the tool to produce predictions that do not match with reality either.

Similarly to the initial interviews, this ground truth was gathered by interaction with employees at Ericsson. Additionally, also here, the employees were selected based on differences in roles and teams. In total, three employees were involved in creating the ground truth. Two of the employees who took part in creating the ground truth also took part in the interviews described in section 4.1 while one new employee was included.

The employees were given a diagram over a class package of which they were familiar with within their work duties. Additionally, they were provided with a list of the classes in that package. The employees were tasked with, for each class in the package, giving a level of importance that the class has for understanding the package. This level of importance was a number ranging from 1 to 10 where 1 means that the class is not important for understanding the package at all and 10 means that the class is very important for understanding the package. The range from 1 to 10 was selected as it is very detailed but not too detailed for employees to select a satisfactory value. Thus, as it is very detailed, it allows for condensing diagrams to precisely the level of abstraction as any reader of the diagram would want to see.

The first employee had the role of design architect. This employee was given a diagram with 51 classes and a diagram with 33 classes to be rated. Thus, a total of 84 classes were rated. The second employee had the role of developer. One diagram with 39 classes was rated by him. The last employee also had the role of developer and he rated 100 classes. Thus, 84 classes were rated by design architects while 139 were rated by developers.

4.4 Step 4: Analysis of metrics

An analysis of the machine learner was done to address RQ2. The performance of the machine learner was measured in a few different ways. The two common techniques, using the holdout method and using k-fold cross-validation, were used. Additionally, for the metrics, the information gain ratio was calculated to get a performance score of the individual metrics. The ground truth described in section 4.3 was used as data for all techniques.

In the holdout method, the data is split into two sets; a training set and a test set. Two thirds of the ground truth data was used for the training set while the remaining third was used for the test set. Which data points that were used in the training set and which that were used in the test set was randomly selected. The proportion of using two thirds of the data in the test set was chosen as this is a common proportion in the holdout method [21].

For the k-fold cross-validation, the same dataset, the ground truth, was utilised. However, for this method, the data did not need to be split prior to using the technique as this is part of the method itself.

For both the holdout method and cross-validation, the outcome is given as the true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) for each possible class importance value. Additionally, the accuracy is given along with the precision and recall to give better understanding of the results. In the holdout method, the full confusion matrix showing the real and predicted values for each class is given for a separate run.

The precision shows the proportion out of all positives that actually were true positives. Thus, in this case, a high precision means that out of the classes that were predicted to be at this importance value, a lot actually belong there. A low precision shows that many of the classes predicted to be in a group actually belong to other groups. The recall shows the proportion of true positives out of all the positives. Here, a high recall for an importance value means that many of the classes that actually have that importance were predicted to be at this importance. A lower value shows that the classes with this importance were predicted to have a faulty importance rating.

Predicting a high importance value for a class that is not very important is not advantageous, but might not affect the diagram as a whole very much. However, a class that is very important may be needed to fully understand the diagram and thus predicting such a class to not be important would be very bad. Thus, it is desirable to have a high recall, especially for the higher importance values. While a high precision would be advantageous as well, it is not as important.

4.5 Step 5: Validation of Condensed diagrams

The condensed diagrams were validated by comparing how useful they are for understanding a package to the reverse engineered diagram with all classes. This validation addresses RQ3. Similarly to the interviews and the gathering of ground truth, the validation of the tool was done together with employees at Ericsson. Also here, the employees were selected based on differences in roles and teams. In total, five employees were selected. Three of these took part in the interviews described in section 4.1 while two new were introduced.

The selected employees were given three diagrams over a package which they were familiar with through their work duties. One diagram contained all the classes and relations found from the reverse engineering. In the second diagram, 20% of the classes were removed. These classes were predicted to be the least important in the diagram. In the last diagram, 40% of the classes were removed. Similarly, the classes predicted to be least important were removed.

The employees were then tasked with giving a number showing how each of the condensed diagrams performed compared to the reverse engineered diagram when it comes to understandability of the package. This number was on the scale from 1 to 5 where 1 means that the condensed diagrams performed very badly and 5 means that the diagram performed very well. Additionally, the employees were asked what made each of the condensed diagrams good and what made the diagrams bad.

Chapter 5

Result

This chapter contains the results of all steps listed in section 4.

5.1 Gathered opinions

Firstly, the results of the questionnaire are shown. These are followed by the codes that were defined in the analysis of the interviews. After that, a mapping between the codes and metrics is given.

5.1.1 Participant background

As can be seen in figure 5.1 more than five answers were given from the five employees who responded to the questionnaire. This is due to the fact that ones work duties may overlap multiple working roles and thus, selection of multiple roles were allowed. However, only one employee selected multiple roles (Design Architect and Developer) while the others only indicated one role. Furthermore, the three roles taking part in the questionnaire was System Architect, Design Architect and Developers.

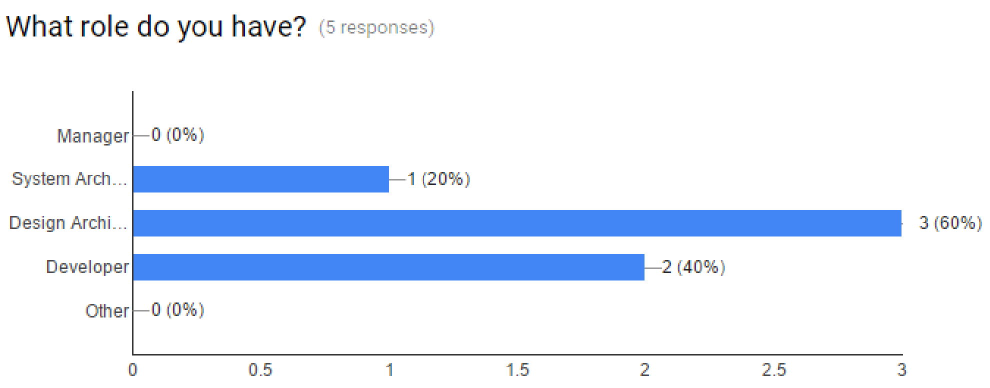

Figure 5.1: The roles of the employees who participated in the study

When it comes to the size of the teams, almost all employees were in teams with six to ten members. The only exception was one employee who were in a smaller team with only one to five members, which is shown in figure 5.2. It should be noted that even though many were in teams of the same size, none of the employees were part in the same team.

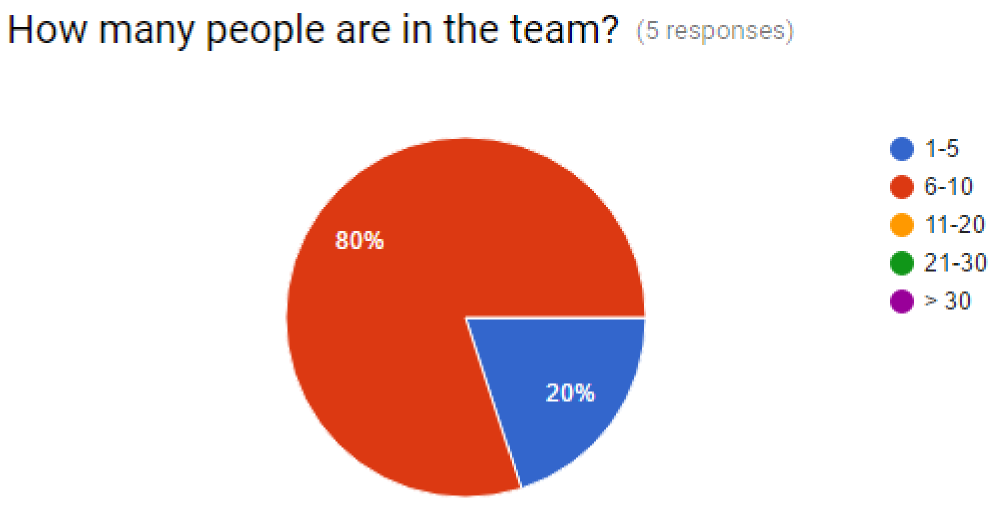

Figure 5.2: The size of the teams the employees who participated in the study belong to

Regarding the prior experience with software development, all of the employees have been working with software for a long time. The questionnaire showed that all employees have over ten years of experience, and this can be seen in figure 5.3.

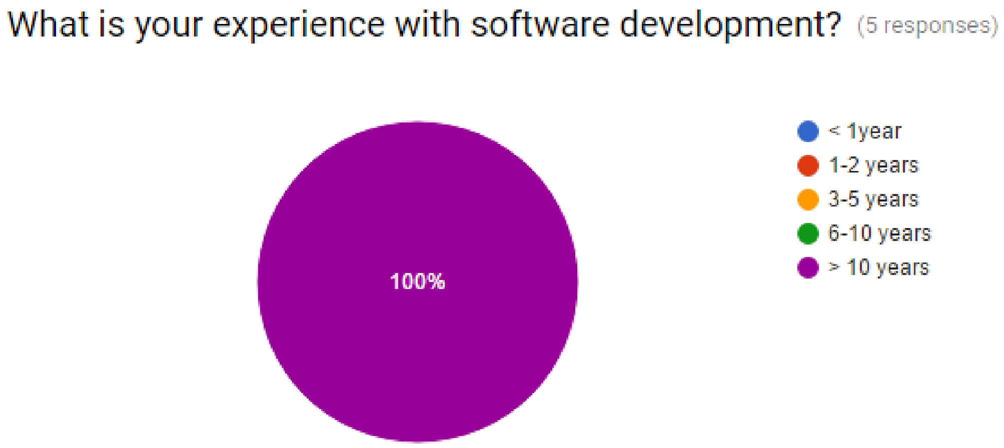

Figure 5.3: Prior experience with software development of the employees who participated in the study

However, for experience with UML, the answers were a bit more diverse. In fact, as shown in figure 5.4, not a single employee gave the same answer as another. Thus, the experience with UML ranges from less than a year to over ten years.

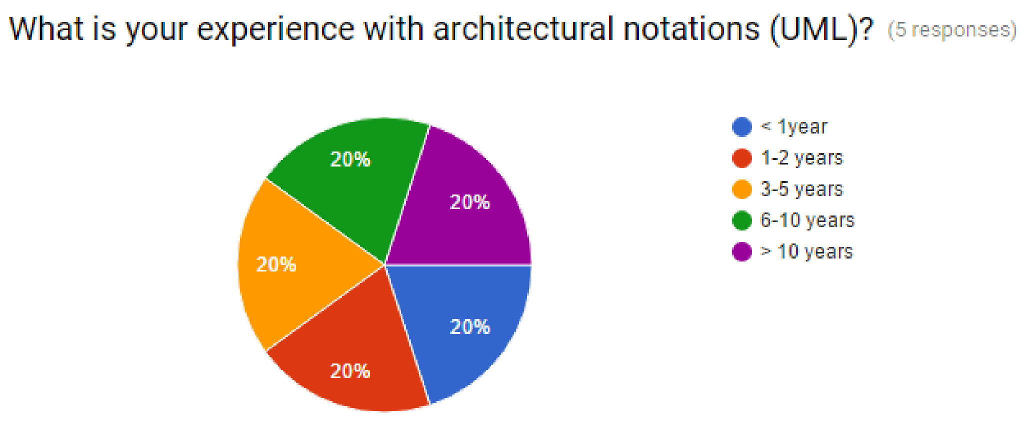

Figure 5.4: Prior experience with UML of the employees who participated in the study

Figure 5.5 shows the diagrams that the employees use. When it comes to the usage of the diagrams, all of the employees use class diagrams regularly. Additionally, almost all employees also use sequence diagrams while a few use state-chart diagrams. However, as figure 5.6 shows, diagrams are not used very often. Though when looking on the usage on scale from one to five, all employees gave responded with at least a two.

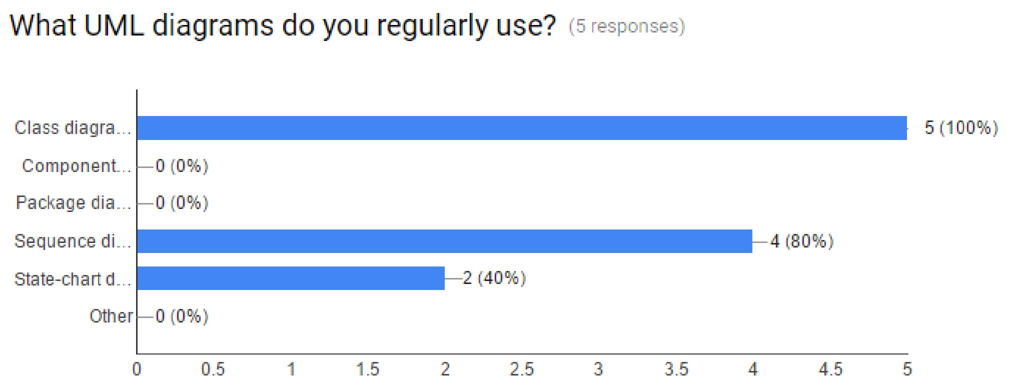

Figure 5.5: The diagram types used by the employees who participated in the study

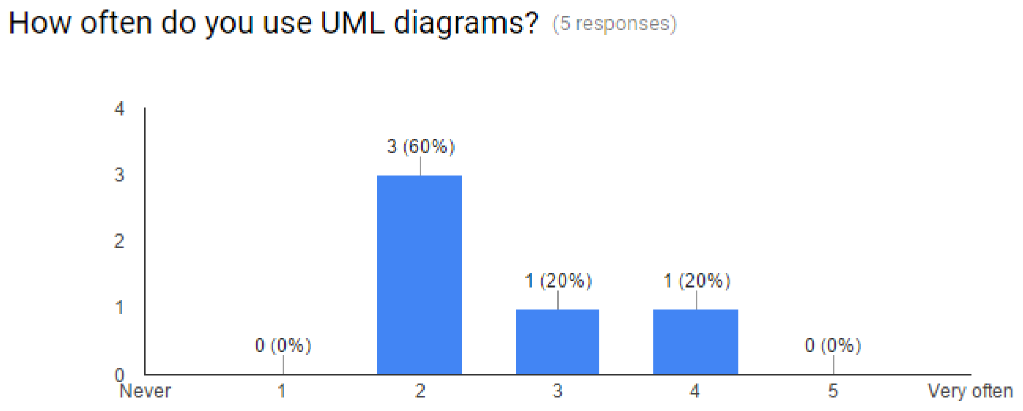

Figure 5.6: The use of diagrams by the employees who participated in the study

5.1.2 Defined codes

Several codes were found to be important in coding process after the interviews. The coding is described in section 4.1.2 and the defined codes are described further below.

Connections

Something that was important within all the stakeholder groups were the connections between different classes. When looking for information in class diagrams, often their goal was to find how classes relate to each other. Some interviewees indicated that the data sent between classes was important for them while others thought the cardinality of the connections mattered. However, the mutual interest was metrics related to the coupling between the classes.

Clusters

A system architect explained that he thought the clustering or grouping of the classes was important for him. What he looked at was how different clusters of classes interacted with other clusters. Furthermore, by seeing clusters, he could analyse the properties of a cluster but also going down on a lower level to look into the cluster when needed.

What he said formed a cluster was mainly due to relations and interfaces. A cluster should not have many relations to classes outside of the clusters. Likewise, there should not be many interfaces that are used from the outside. However, within a cluster, there are a lot less restrictions on how the relations and interfaces are placed. A lot of relations between classes as well as shared use of data was allowed within the cluster.

A design architect agreed that clusters were meaningful as well. However, he explained that in a cluster with high collaboration there were likely to exist some important classes but some unimportant help classes as well.

Metrics that are relevant to these clusters are thus related to the coupling between classes, but also to the available functions within the classes.

Erroneous classes

Classes that were affected by errors such as having bugs or breaking rules and conventions were considered as more important for all stakeholders. They explained that such classes could be an indication of a larger problem where a fix or a redesign would be needed.

While one of the developers agreed that this erroneous classes were important because of the effect they have on the system, he added that it is less important in his role as developer as it would be in a role on a higher level such as an architect.

Any metric related to faults in classes is relevant here. These could be metrics such as the number of bugs in the system or the number of changes that have been made in the system.

Complexity

Complexity is something that was considered as important mainly for developers. A developer said that this is because if a change is needed to be done in a complex class, it is more difficult to implement the change, thus making the complex classes more important for him in his work.

A developer explained that classes can be rather large but still not contain a lot of logic. These classes could include a lot of boilerplate code and mostly be used to send data back and forth. They could also be very repetitive in what they do. While these classes with little logic still can be large, they are often self explanatory and it is usually easy to understand them without looking at them in more detail.

Additionally, one design architect added that complexity is not always important. An example of this was given with a class that had very specific functionality. This class had a high complexity, but was only used in some specific situation which rarely occurred. Such a class was therefore not considered as important within the system while still having a high complexity.

Another design architect pointed out that tools measuring complexity already are being used to find faults in the software. However, it rarely helped him in his work and thus he disagreed with complexity making a class more important.

Volatility

Classes that are changed a lot are more important according to each of the stakeholder groups. Being changed a lot could be an indication of that something is wrong, and thus the design could be subject to change due to this.

However, one of the design architects also stated that there are some classes that are core to the system that many changes go through. While these classes are not showing any problems, they are still changed a lot and are considered as important within the system.

Furthermore, another design architect mentioned that it is important for him to keep track of the classes that are changed a lot to make sure that they are maintained, even if there is no sign of error.

Change impact

A class that, when changed, has a high impact so that it may affect many other classes in a way that the affected classes also need some kind of change is something that was considered as important for all stakeholder groups.

As these classes may affect many other, it is necessary to keep them well maintained.

However, one developer explained that he was working on such a low level where he needed to decide exactly what was needed to change for a feature to be developed. Therefore, knowing that a class has a high impact did not help him in his work as he already knew all the needed changes. Thus, he did not consider such classes important.

Functionality

Both developers and design architects indicated that it is what type of functionality a class is involved in that makes a class important. Classes that are involved in the main functionality of a system are more important than classes that are only used in some specific situation. Such an important functionality could be starting up the system. On the contrary, a class which is only used to perform a change in settings of the software is less important.

5.1.3 Mapping - Codes and metrics

The metrics that were available were mapped with the identified codes. This mapping can be seen in figure 5.7. As can be seen, both Connections and Clusters regard various design metrics. At the same time, neither Change impact nor Functionality were related to any of the metrics that were available.

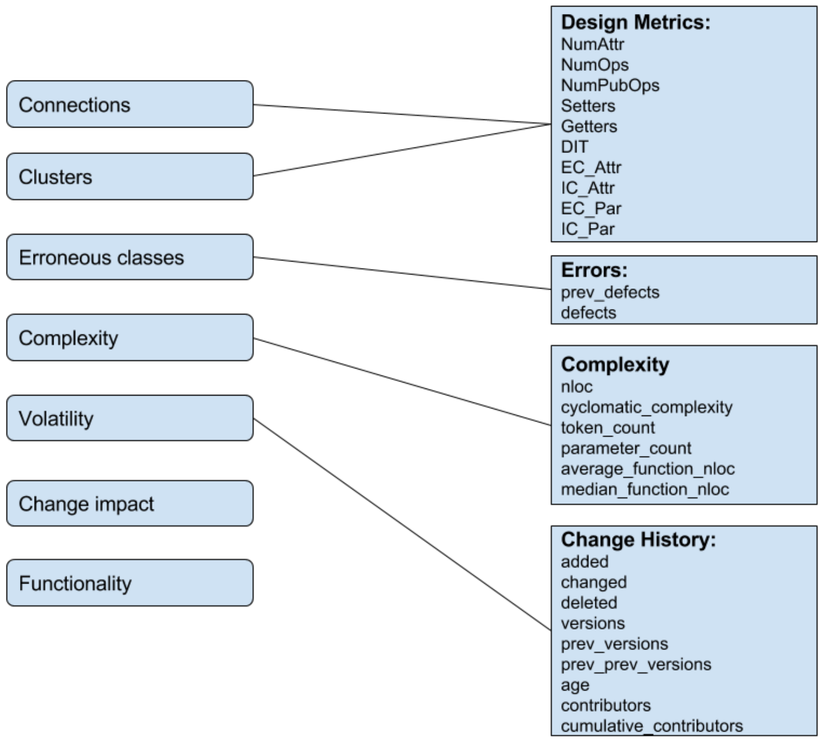

Figure 5.7: The codes found through the interviews mapped with the available metrics.

5.2 Performance of metrics

Both in the holdout method and the cross validation, the outcome is given. As there are ten possible values of the importance of a class and not a simple true or false, the outcome is given for all ten class importance values. Furthermore, the distribution of the ground truth which is used as input in the analysis can be seen in figure 5.8. However, 53 of these classes did not have metrics available and were thus disregarded from the analysis and 170 classes remained.

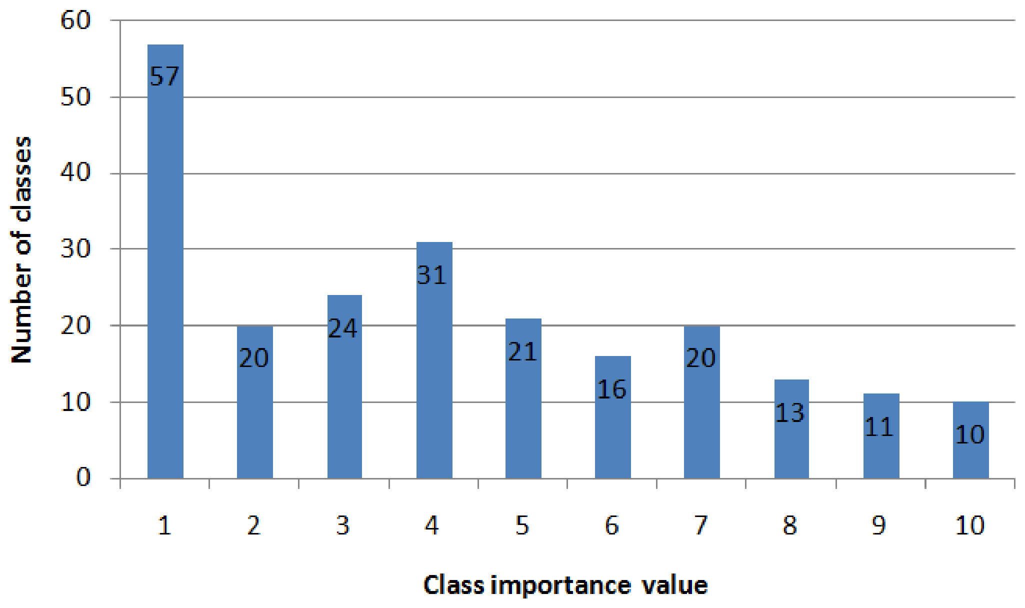

Figure 5.8: A histogram showing the distribution of the classes rated in the ground truth.

5.2.1 Holdout method

The holdout method, which is described in section 4.4, was used with all different groups of metrics. Additionally, the method was used with all metrics together.

All metrics

Figure 5.1 shows the result when all metrics were included. Generally speaking, all class values have a high accuracy. It is larger than 0.8 in all cases other than for class value 3. However, the precision is consistently lower than the accuracy. For many class values, the precision is 0, which means that there were no true positives. However, it spikes for some class values such as class value 1 and class value 9 where it reaches a precision of 0.8750 and 0.7500 respectively. Similarly, the recall is very high for class value 1 and then has a few spikes while it also goes to 0 for some class values.

From the outcome, it can be seen that there are a lot of true negatives for all class values. Class value 1 has a lot of true positives while the others remain at very few. False negatives and false positives are less than 10 for each of the class values.

Class Value	Accuracy	Precision	Recall	Outcome
1	0.9649	0.8750	1.0000	[TP=14, FP=2, TN=41, FN=0]
2	0.8772	0.3333	0.4000	[TP=2, FP=4, TN=48, FN=3]
3	0.7544	0.1000	0.1667	[TP=1, FP=9, TN=42, FN=5]
4	0.8421	0.3750	0.4286	[TP=3, FP=5, TN=45, FN=4]
5	0.8421	0.0000	0.0000	[TP=0, FP=6, TN=48, FN=3]
6	0.9649	0.0000	0.0000	[TP=0, FP=1, TN=55, FN=1]
7	0.8421	0.0000	0.0000	[TP=0, FP=1, TN=48, FN=8]
8	0.9474	0.0000	0.0000	[TP=0, FP=2, TN=54, FN=1]
9	0.8947	0.7500	0.3750	[TP=3, FP=1, TN=48, FN=5]
10	0.8772	0.0000	0.0000	[TP=0, FP=3, TN=50, FN=4]

Table 5.1: Results from holdout method with all metrics.

The confusion matrix in table 5.2 shows that there were a lot of classes with importance value 1 that also were predicted to be 1. However, some classes were predicted to be far form their actual values. This includes two classes with importance value one which were predicted to be 9 and 10 respectively. Additionally, some classes with importance value 8 were predicted to be lower than 4.

		Predicted									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Actual value	**1**	11	0	0	0	0	0	0	0	0	0
	2	2	1	3	0	0	0	0	0	0	0
	3	1	0	2	1	0	0	1	0	0	0
	4	1	0	1	2	1	0	0	0	1	1
	5	0	0	2	0	0	0	2	0	0	0
	6	1	0	4	1	0	0	0	0	0	0
	7	0	0	0	2	0	0	0	0	1	0
	8	2	2	2	1	0	0	0	0	0	0
	9	0	2	1	1	0	1	0	0	1	0
	10	0	0	0	1	1	0	0	0	0	0

Table 5.2: Confusion matrix from one run of the holdout method with all metrics.

Design metrics

Similarly to when using all metrics, the design metrics have an accuracy where most values are above 0.8 while the precision is low with a few exceptions. However, as can be seen in table 5.3, Class value 1 which only had 2 false positives with all metrics increased to 14 false positive when using the design metrics only. Thus, the accuracy of class value 1 decreased significantly from the previous table.

Class Value	Accuracy	Precision	Recall	Outcome
1	0.6842	0.4167	0.7143	[TP=10, FP=14, TN=29, FN=4]
2	0.8070	0.0000	0.0000	[TP=0, FP=9, TN=46, FN=2]
3	0.8596	0.0000	0.0000	[TP=0, FP=5, TN=49, FN=3]
4	0.7895	0.3077	0.5714	[TP=4, FP=9, TN=41, FN=3]
5	0.8947	0.0000	0.0000	[TP=0, FP=1, TN=51, FN=5]
6	0.9123	NaN	0.0000	[TP=0, FP=0, TN=52, FN=5]
7	0.8947	0.0000	0.0000	[TP=0, FP=1, TN=51, FN=5]
8	0.8772	0.0000	0.0000	[TP=0, FP=1, TN=50, FN=6]
9	0.8947	0.0000	0.0000	[TP=0, FP=2, TN=51, FN=4]
10	0.8772	0.0000	0.0000	[TP=0, FP=1, TN=50, FN=6]

Table 5.3: Results from holdout method with metrics related to the design.

The confusion matrix in table 5.4 is similar to previous confusion matrix where many classes with importance of 1 were predicted to be 1.

		Predicted									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	1	12	1	0	0	2	0	0	0	1	0
	2	2	1	0	0	0	0	0	0	0	0
	3	0	3	2	0	0	0	0	0	0	1
	4	0	1	0	3	1	0	0	0	0	1
Actual	**5**	0	1	3	2	1	0	0	0	0	0
value	**6**	0	1	0	0	0	0	0	0	0	0
	7	1	0	1	2	2	0	0	0	1	0
	8	0	1	1	1	1	0	1	0	0	0
	9	0	0	0	0	0	0	0	0	1	0
	10	1	3	0	1	0	0	0	0	0	0

Table 5.4: Confusion matrix from one run of the holdout method with only design metrics.

Complexity

Table 5.5 shows a similar trend with high accuracy values and a precision at 0 for most class values. Additionally, class value 1 has 16 false positives which is the largest number of false positives in all metric groups.

Class Value	Accuracy	Precision	Recall	Outcome
1	0.6842	0.4285	0.8571	[TP=12, FP=16, TN=27, FN=2]
2	0.8596	0.0000	0.0000	[TP=0, FP=2, TN=49, FN=6]
3	0.8070	0.0000	0.0000	[TP=0, FP=5, TN=46, FN=6]
4	0.8421	0.2500	0.4000	[TP=2, FP=6, TN=46, FN=3]
5	0.9298	0.7500	0.5000	[TP=3, FP=1, TN=50, FN=3]
6	0.8947	0.0000	0.0000	[TP=0, FP=1, TN=51, FN=5]
7	0.9123	0.0000	0.0000	[TP=0, FP=2, TN=52, FN=3]
8	0.9649	NaN	0.0000	[TP=0, FP=0, TN=55, FN=2]
9	0.9123	0.4000	0.5000	[TP=2, FP=3, TN=50, FN=2]
10	0.8596	0.0000	0.0000	[TP=0, FP=2, TN=49, FN=6]

Table 5.5: Results from holdout method with metrics related to the complexity.

For table 5.6, there are some bad predictions such as classes with importance of 2 being predicted as 9 and 10 and a class with importance of 9 being predicted to be 1.

		Predicted									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	1	8	0	7	0	0	0	1	0	0	0
	2	1	0	2	0	0	1	0	0	1	2
	3	1	0	0	1	1	0	0	0	0	0
	4	1	0	3	1	0	1	0	0	0	1
Actual value	**5**	1	0	3	1	1	0	0	0	0	0
	6	0	0	2	0	0	0	0	0	0	0
	7	1	0	2	0	2	0	0	0	0	0
	8	0	0	3	1	0	1	0	0	1	0
	9	1	0	1	0	0	1	0	0	1	0
	10	0	0	0	0	0	0	0	0	1	0

Table 5.6: Confusion matrix from one run of the holdout method with only complexity metrics.

Change history

Also when it comes to change history metrics, it is possible to see form table 5.7 that the same trends exist. However, in this case, class value 3 had a lot of false positives rather than class value 1.

Class Value	Accuracy	Precision	Recall	Outcome
1	0.7895	0.4167	0.5000	[TP=5, FP=7, TN=40, FN=5]
2	0.8947	0.5000	0.3333	[TP=2, FP=2, TN=49, FN=4]
3	0.6842	0.0000	0.0000	[TP=0, FP=12, TN=39, FN=6]
4	0.8070	0.2857	0.2500	[TP=2, FP=5, TN=44, FN=6]
5	0.8947	NaN	0.0000	[TP=0, FP=0, TN=51, FN=6]
6	0.9298	0.0000	0.0000	[TP=0, FP=2, TN=53, FN=2]
7	0.8421	0.0000	0.0000	[TP=0, FP=4, TN=48, FN=5]
8	0.8070	0.1250	0.2000	[TP=1, FP=7, TN=45, FN=4]
9	0.9298	0.6250	0.8333	[TP=5, FP=3, TN=48, FN=1]
10	0.9474	NaN	0.0000	[TP=0, FP=0, TN=54, FN=3]

Table 5.7: Results from holdout method with metrics related to the change history.

Looking at table 5.8, there are, just as previously, a lot of classes with importance of 1 being predicted as 1, while there are some large mispredictions.

		Predicted									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	1	12	1	0	1	0	0	0	0	1	0
	2	1	1	3	0	0	0	0	0	0	0
	3	3	3	2	0	0	0	1	0	0	0
	4	0	4	1	1	0	0	0	0	1	0
Actual value	**5**	3	3	0	0	0	0	1	0	0	0
	6	1	1	0	1	0	0	0	0	0	0
	7	1	2	1	1	0	0	0	0	0	0
	8	2	1	0	0	0	0	0	0	0	0
	9	0	0	0	0	0	0	0	0	1	0
	10	0	1	0	0	1	0	0	0	0	0

Table 5.8: Confusion matrix from one run of the holdout method with only change history metrics.

Error

All the data points for the metrics in the error group had the value 0. Calculating the nearest neighbor when all data lie within the same point is not practical and thus this group was disregarded for this test.

Comparison - holdout method

It is clear that for class importance value 1, using all metrics produced the best result. A precision of 0.8750 and a recall of 1.0000 was reached while both values were a lot lower for the other metric groups. For class value 2, both design metrics and complexity metrics score 0 for both precision and recall, however, change history manages to reach a precision of 0.5000 and a recall of 0.3333. Design metrics only seem to perform well for class value 1 and 4. Complexity metrics perform well for importance values 1, 4, 5 and 9 but gets values of 0 for the others. Change history metrics reach non-zero values for importance 1, 2, 4, 8 and 9, however, for class value 8, the precision is as low as 0.1250 and the recall is 0.8333.

Predictions in many runs

When the holdout method was used 1000 times, the results, which can be seen in table 5.9, were acquired. 1000 runs were used as the result was a bit varying between the different runs.

Metric group	Accuracy	Precision	Recall
All	0.8644	0.3427	0.3423
Design	0.8605	0.3217	0.3214
Complexity	0.8529	0.2814	0.2810
Change history	0.8501	0.2694	0.2690

Table 5.9: The results of 1000 runs for each of the metric groups.

Overall, using all metrics gave the best results with the highest accuracy, precision and recall. Design was just a little bit lower for all three values. Complexity and change history metrics score the lowest.

5.2.2 Cross validation

Similarly to the holdout method, the cross validation method was used with all different groups of metrics and with all metrics together. The cross validation was performed with 10 folds.

All metrics

When it comes to the result, the cross validation reached similar values to the holdout method which can be seen in table 5.10. Thus, it had an accuracy where most values are high and a precision and recall which are low with a few exceptions.

Class Value	Accuracy	Precision	Recall	Outcome
1	0.8941	0.6875	0.9167	[TP=33, FP=15, TN=119, FN=3]
2	0.8000	0.1538	0.2500	[TP=4, FP=22, TN=132, FN=12]
3	0.8000	0.2069	0.3529	[TP=6, FP=23, TN=130, FN=11]
4	0.7882	0.2414	0.3333	[TP=7, FP=22, TN=127, FN=14]
5	0.8471	0.0909	0.0588	[TP=1, FP=10, TN=143, FN=16]
6	0.9000	0.1429	0.0833	[TP=1, FP=6, TN=152, FN=11]
7	0.8647	0.0000	0.0000	[TP=0, FP=6, TN=147, FN=17]
8	0.9235	0.5000	0.0769	[TP=1, FP=1, TN=156, FN=12]
9	0.9412	0.5714	0.3636	[TP=4, FP=3, TN=156, FN=7]
10	0.9118	0.0000	0.0000	[TP=0, FP=5, TN=155, FN=10]

Table 5.10: Results from cross-validation with all metrics.

Design metrics

The results for the design metrics can be seen in table 5.11. Also here, there are a lot of true positives for class value 1 while the others remain lower.

Class Value	Accuracy	Precision	Recall	Outcome
1	0.7882	0.5000	0.8055	[TP=29, FP=29, TN=105, FN=7]
2	0.8176	0.2222	0.3750	[TP=6, FP=21, TN=133, FN=10]
3	0.8471	0.2857	0.3529	[TP=6, FP=15, TN=138, FN=11]
4	0.7765	0.2424	0.3810	[TP=8, FP=25, TN=124, FN=13]
5	0.8471	0.0909	0.0588	[TP=1, FP=10, TN=143, FN=16]
6	0.9294	NaN	0.0000	[TP=0, FP=0, TN=158, FN=12]
7	0.8882	0.0000	0.0000	[TP=0, FP=2, TN=151, FN=17]
8	0.9176	0.0000	0.0000	[TP=0, FP=1, TN=156, FN=13]
9	0.9235	0.3750	0.2727	[TP=3, FP=5, TN=154, FN=8]
10	0.9000	0.1111	0.1000	[TP=1, FP=8, TN=152, FN=9]

Table 5.11: Results from cross-validation with metrics related to design.

Complexity

Results for complexity metrics are shown in table 5.12 and follow a similar trend.

Class Value	Accuracy	Precision	Recall	Outcome
1	0.6824	0.3750	0.7500	[TP=27, FP=45, TN=89, FN=9]
2	0.9000	0.4000	0.1250	[TP=2, FP=3, TN=151, FN=14]
3	0.8471	0.1538	0.1176	[TP=2, FP=11, TN=142, FN=15]
4	0.7647	0.1935	0.2857	[TP=6, FP=25, TN=124, FN=15]
5	0.8647	0.2857	0.2353	[TP=4, FP=10, TN=143, FN=13]
6	0.9059	0.0000	0.0000	[TP=0, FP=4, TN=154, FN=12]
7	0.8706	0.1429	0.0588	[TP=1, FP=6, TN=147, FN=16]
8	0.9118	0.0000	0.0000	[TP=0, FP=2, TN=155, FN=13]
9	0.9118	0.3571	0.4545	[TP=5, FP=9, TN=150, FN=6]
10	0.8941	0.0000	0.0000	[TP=0, FP=8, TN=152, FN=10]

Table 5.12: Results from cross-validation with metrics related to complexity.

Change history

Also for change history metrics, the trend can be observed and this is seen in table 5.13

Class Value	Accuracy	Precision	Recall	Outcome
1	0.7765	0.4815	0.7222	[TP=26, FP=28, TN=106, FN=10]
2	0.8647	0.2941	0.3125	[TP=5, FP=12, TN=142, FN=11]
3	0.7647	0.1034	0.1765	[TP=3, FP=26, TN=127, FN=14]
4	0.8059	0.2500	0.2857	[TP=6, FP=18, TN=131, FN=15]
5	0.8765	0.0000	0.0000	[TP=0, FP=4, TN=149, FN=17]
6	0.8941	0.0000	0.0000	[TP=0, FP=6, TN=152, FN=12]
7	0.8647	0.0000	0.0000	[TP=0, FP=6, TN=147, FN=17]
8	0.8294	0.0556	0.0833	[TP=1, FP=17, TN=140, FN=12]
9	0.9176	0.3636	0.3636	[TP=4, FP=7, TN=152, FN=7]
10	0.9353	0.0000	0.0000	[TP=0 FP=1, TN=159, FN=10]

Table 5.13: Results from cross-validation with metrics related to change history.

Error

As the same data was used here as in the holdout method, all values were 0 and this group was disregarded in the test.

Comparison - crossvalidation

The precision and recall does not as often go as low as zero for most of the class values as it does in the holdout method. However, it is clear that the different groups perform differently for the various class importance values. Similar to in the holdout method, using all metrics greatly outperforms the smaller metric groups for class importance value 1. For importance values other than 1, the design metrics perform their best and reach their highest values of recall at importance 2, 3 and 4. Additionally, the highest precision is reached for importance 9. Complexity metrics, however, reach a high recall for class value 1 and 9, but not very high for the others. When it comes to change history metrics, importance values 1, 2, 4 and 9 reach the highest values of recall and precision. However, 5, 6, 7 and 10 are all zero in both recall and precision. Thus, some class importance values do not reach good results for any metrics groups. Such importance values are 5, 6, 7, 8 and 10.

5.2.3 Feature scoring

When it comes to the performance of each individual metric, it can be seen in table 5.14 that IC_Par and EC_Par which both are design metrics score the highest. These are followed by token_count and cyclomatic_complexity which are located in the complexity group. Afterwards come the metrics changed and added which both are related to the change history.

In the bottom of the table, however, are the metrics prev_defects, defects, prev_versions and prev_prev_versions which all get a score of 0.

Metric Group	**Name**	**Information gain ratio**
Design Metrics	IC_Par	0.4246
Design Metrics	EC_Par	0.3901
Complexity	token_count	0.3254
Complexity	cyclomatic_complexity	0.2684
Change History	changed	0.2669
Change History	added	0.2599
Design Metrics	DIT	0.2559
Complexity	parameter_count	0.2470
Design Metrics	EC_Attr	0.2456
Complexity	nloc	0.2411
Complexity	average_function_nloc	0.2356
Design Metrics	IC_Attr	0.2250
Design Metrics	nrOp	0.2062
Change History	deleted	0.2032
Design Metrics	nrGetter	0.1990
Design Metrics	nrSetter	0.1894
Change History	cumulative_contributors	0.1882
Complexity	median_function_nloc	0.1846
Design Metrics	nrAttr	0.1820
Design Metrics	nrPubOp	0.1795
Change History	versions	0.1587
Change History	contributors	0.1450
Change History	age	0.1127
Errors	prev_defects	0.0000
Errors	defects	0.0000
Change History	prev_versions	0.0000
Change History	prev_prev_versions	0.0000

Table 5.14: Information gain ratio of each metric that was included.

5.3 Performance of condensed diagrams

All of the diagrams displayed in this section have had their names changed due to confidentiality of the software. Each diagram has had their elements being renamed separately. Thus, a class named *Class_1* in one diagram may be a completely different class than a class with the same name in another diagram.

5.3.1 Diagram 1

Figure 5.9 shows the diagram made out of all the classes of a package after the reverse engineering process of the source code. This diagram along with the diagram in figure 5.10 and figure 5.11 was sent to one of the employees.

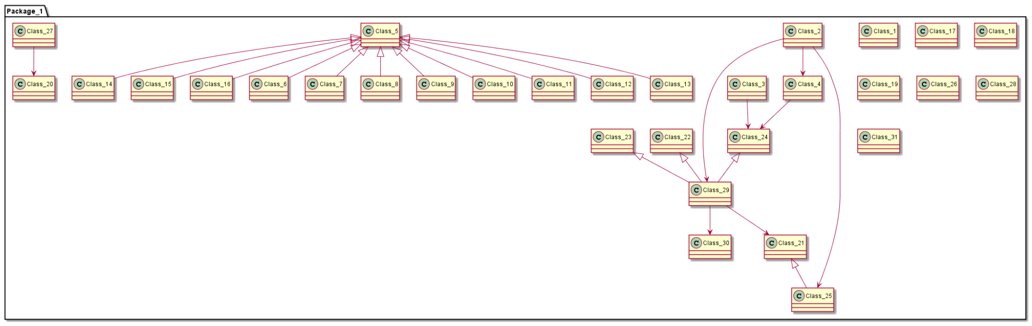

Figure 5.9: Full reverse engineered package diagram with all classes remaining.

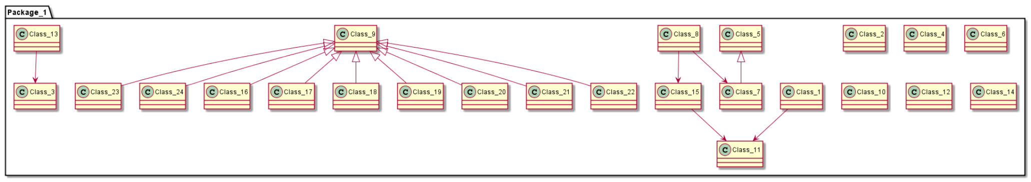

Figure 5.10: Reverse engineered package diagram with the 80% most important classes remaining.

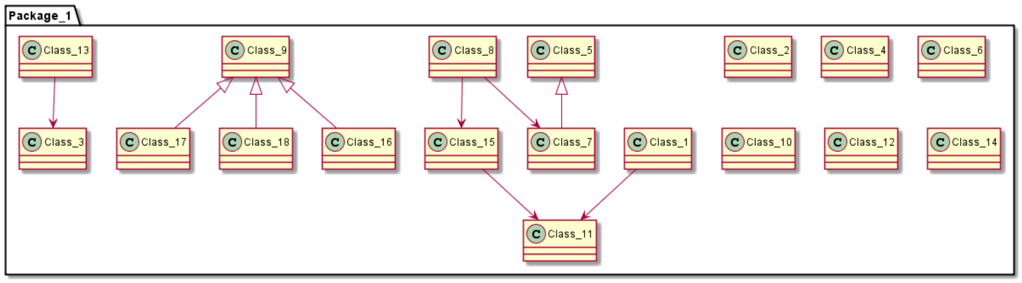

Figure 5.11: Reverse engineered package diagram with the 60% most important classes remaining.

On the scale from one to five of how good the diagram is for understanding the package, the employee gave figure 5.10 a four. He gave the reason that almost all important classes were shown in the diagram while the less important were excluded. However, there was one specific class which he considered important that had been removed.

Similarly to the previous diagram, the figure 5.11 was given the score four. The same reason was given and also here, one important class was removed in the condensing process.

5.3.2 Diagram 2

Diagram 2, which is shown in figure 5.12 is significantly larger than diagram 1. This diagram was also sent to one employee along with the two condensed versions.

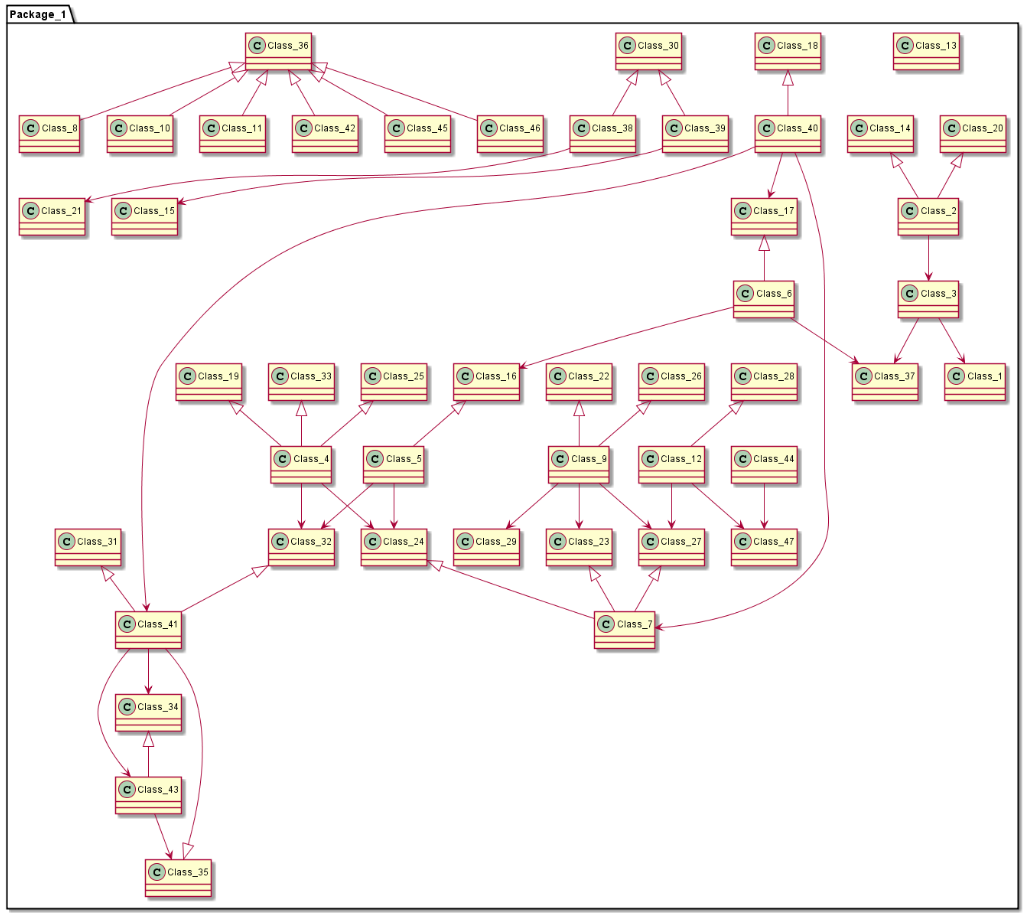

Figure 5.12: Full reverse engineered package diagram with all classes remaining.

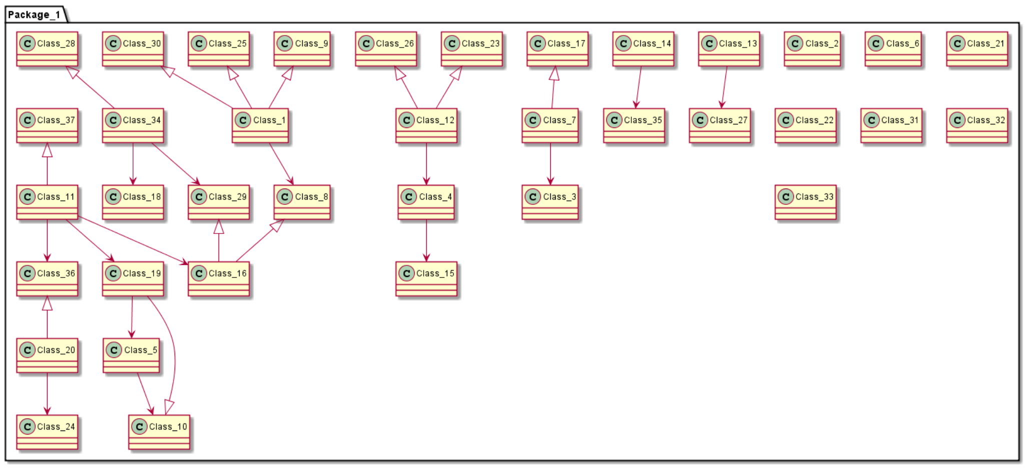

Figure 5.13: Reverse engineered package diagram with the 80% most important classes remaining.

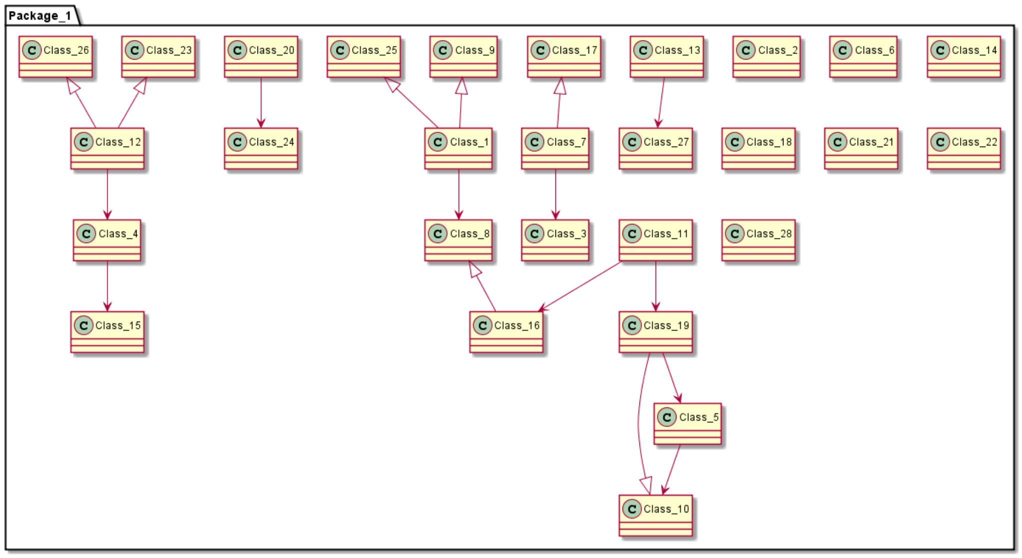

Figure 5.14: Reverse engineered package diagram with the 60% most important classes remaining.

The diagram shown in figure 5.13 was rated 3.5 on the scale from 1 to 5 by the employee. The employee indicated that this was due to the fact that it had fewer classes than the original version.

The same employee gave the diagram in figure 5.14 a four. Similarly to the previous diagram, the fact that it had fewer classes made was positive in the employees opinion. Additionally, the increase of space between classes made it easier to see the structure of the package.

Although both condensed diagrams increased the understandability of the package compared to the original diagram, the employee stated that he would need more information to fully understand the package. It was indicated that a note with a brief explanation of the behaviour of the package would be useful for understandability. Additionally, a label on a specific dependency between two classes showing that one creates the other would be very helpful in this specific case.

5.3.3 Diagram 3

Figure 5.15 shows the smallest diagram out of all the diagrams in the validation. This diagram was also send to one employee along with the condensed diagrams.

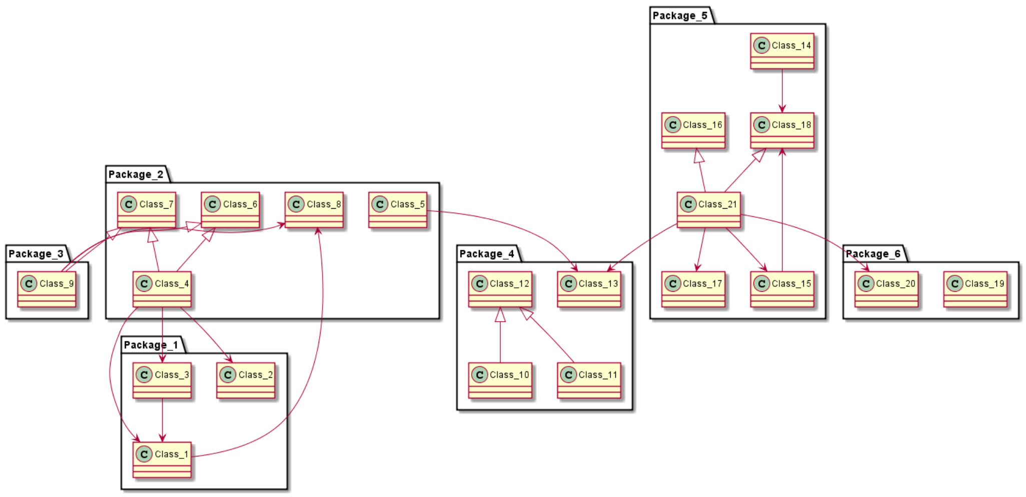

Figure 5.15: Full reverse engineered package diagram with all classes remaining.

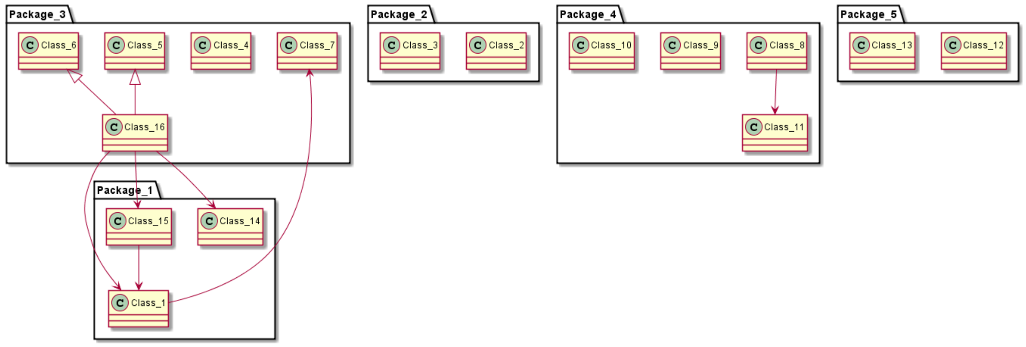

Figure 5.16: Reverse engineered package diagram with the 80% most important classes remaining.

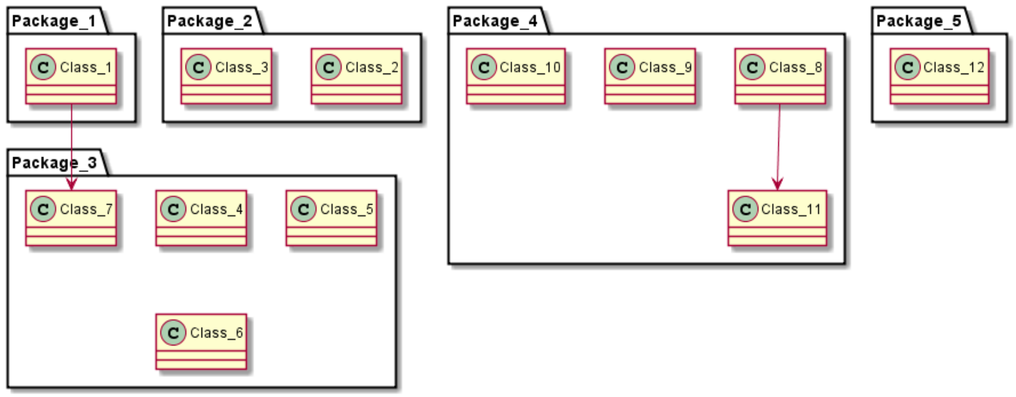

Figure 5.17: Reverse engineered package diagram with the 60% most important classes remaining.

The diagram shown in 5.16 was rated one on the scale from one to five. The employee explained that not much information could be retrieved from the diagram and that some of the most important classes were removed from the diagram.

Similarly to the previous diagram, the diagram in 5.17 was also rated one due to lack of information and important classes being removed. Additionally, the lack of relations in the diagram was not well appreciated.

The employee explained that relations between classes generally never should be removed. Additionally, the employee felt that the original reverse engineered diagram was good at the current scope. It was indicated that condensation of diagrams should only be done when there are a lot more classes where orientation is not as manageable.

5.3.4 Diagram 4

The diagram showed in figure 5.18 was given to two employees to be rated, however, separately from each other.

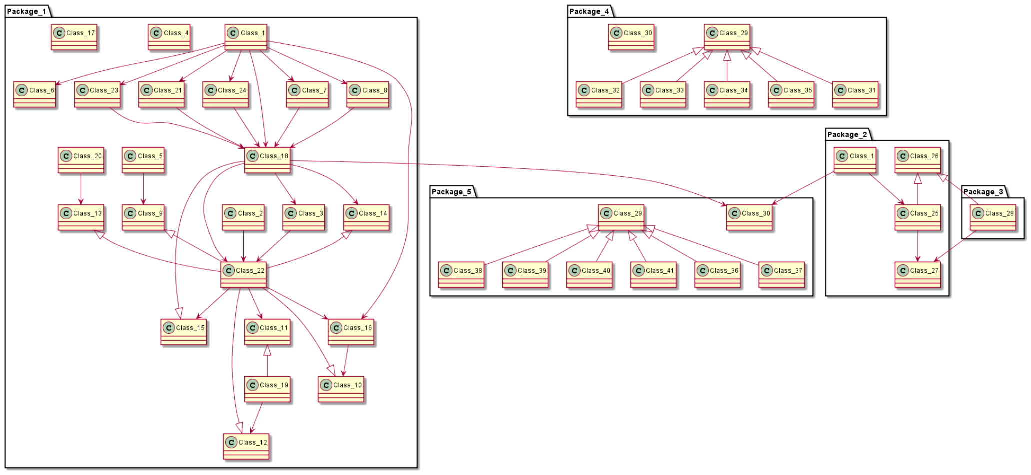

Figure 5.18: Full reverse engineered package diagram with all classes remaining.

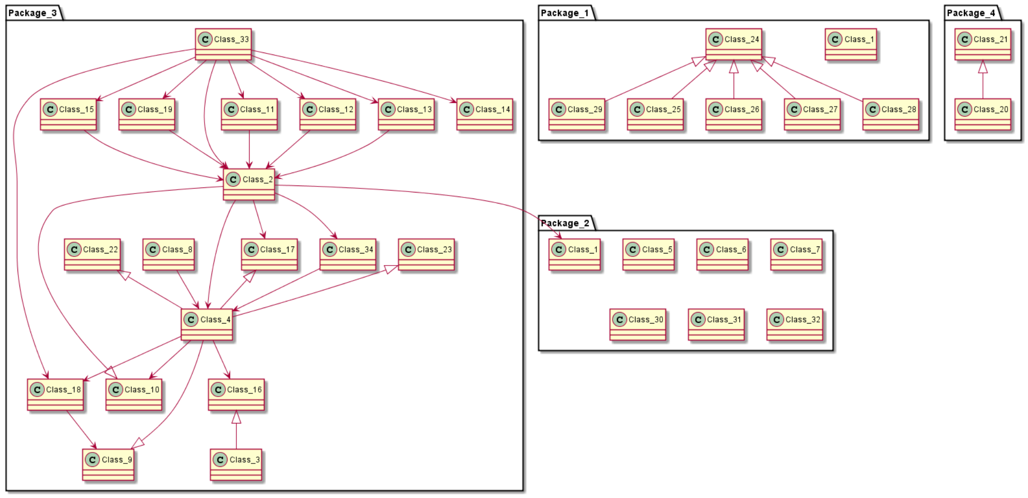

Figure 5.19: Reverse engineered package diagram with the 80% most important classes remaining.

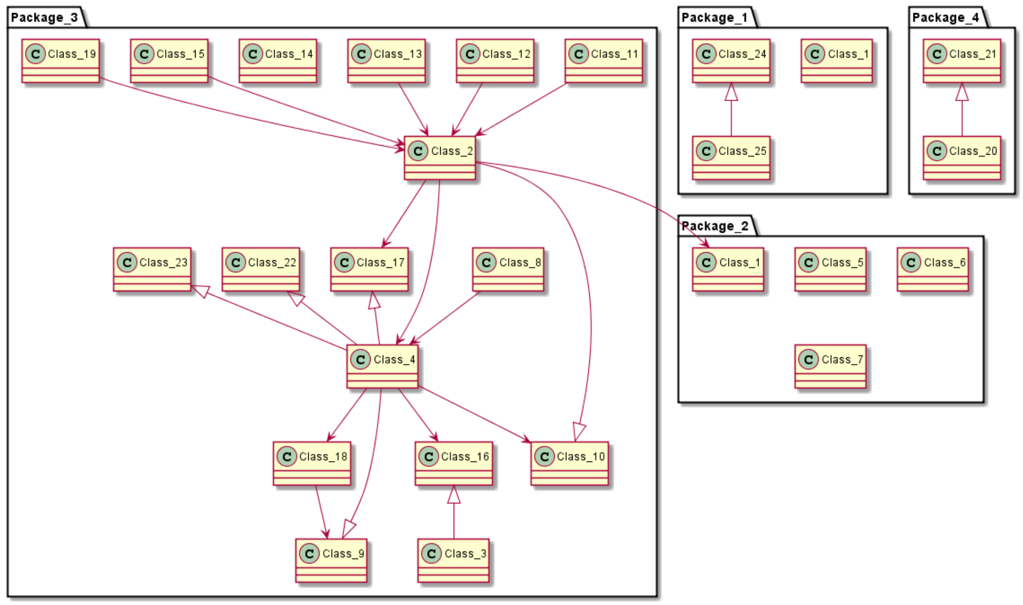

Figure 5.20: Reverse engineered package diagram with the 60% most important classes remaining.

The first employee rated the diagram in figure 5.19 to be a four. However it was indicated that one of the sub-packages is missing a bit of information and could be removed. The second employee gave the same diagram a rating of 3. This employee stated that the layout was improved in the condensed diagrams, and it became easier to follow the lines. However, there seemed to be a lack of relations, especially in the case when the relation was using an *auto_pointer*. Additionally, classes that are not inside this package, but still have relations to classes in the package would be useful to include. This is to fully understand the purpose of classes inside the package.

For the diagram shown in figure 5.20, the first employee gave the rating of four as well. He explained that the largest sub-package was better than in the previous diagram, being more condensed while still showing interesting information. However, two of the smaller packages had lost information was was not considered to be less important. The second employee also rated this diagram with a four. However, just as in the previous diagram, the second employee saw the lack of some classes and relations as negative.

5.3.5 Ratings of all diagrams

All of the ratings for the diagrams shown in the previous sections can be seen in table 5.15. From the table, it is clear that the rating is at least as high when there is just 60% of the classes left as when there is 80% of the classes left. In all cases except for one diagram, the rating is at least 3, which shows that the condensed diagrams performed at least as well as the original reverse engineered diagram.

	Condensed (80% remaining)	Condensed (60% remaining)
Diagram 1	4	4
Diagram 2	3.5	4
Diagram 3	1	1
Diagram 4 (Employee 1)	4	4
Diagram 4 (Employee 2)	3	4
Average	**3.1**	**3.4**

Table 5.15: Ratings of the condensed diagrams.

Chapter 6

Discussion

In this section, first the results are discussed. This is followed by threats that may affect validity and future works to build further on the findings.

6.1 Gathered opinions

The background questions gave insight in the previous experience of the involved interviewees. Something that was consistent between all the interviewees was that all had a very thorough experience with software development. In fact, all the interviewees had been part of the industry for at least ten years. For the interviews, this was very helpful as they could give rigorous answers in the interviews and elaborate on why something was seen as important for them. However, while the experience with UML was a bit more varying and not as rigorous for all interviewees, all interviewees were still familiar with and regularly used class diagrams and could thus still provide valuable feedback.

From the interviews, several codes were defined. These codes represent what makes a class important for the interviewees and the various information needs they have from the software. While these are not direct metrics that can be used within the machine learner, they give an indication of the type of metrics that are important. Thus, it was possible to successfully link the codes with with the metric groups and then use these groups within the machine learner.

While many interviewees had the same opinion, that most aspects were considered very important for them, they put slightly more emphasis on some aspects over others. Additionally, there were some instances were opinions from different interviewees opposed each other. While differences in roles contributed towards different opinions in some cases, it still happened that interviewees with the same role disagreed with one other. Whether differences in working tasks affected these disagreements or not was never determined.

RQ1: Which metrics of a software class are considered to be architecturally important for different stakeholders within the software development?
Metrics related to connections between classes, clusters of classes, errors in classes, complexity of classes, volatility of classes, change impact of classes and functionality of classes are considered architecturally important for all stakeholders.

6.2 Performance of metrics

When looking at all of the confusion matrices in section 5.2, it is clear that there are a lot of true negatives for all difference class important values. This is because one class is predicted to have just 1 out of the 10 possible importance ratings. Thus, even if a class is put in the wrong group, eight of the groups would still count the class as a true negative. This high amount of true negatives may cause the accuracy to look very high.

However, it should still be noted that a prediction that is wrong is necessarily not bad. For example, a class which in reality has the importance rating 9 would be considered to be very important. However, if it was predicted that this class only had the importance 8, it would be considered an incorrect classification. But the value 8 is very close to 9, and even though the class was wrongly classified, it would still be seen as very important.

Having ten different importance values allowed for being very precise when selecting the size of the condensed diagrams. Putting the classes into fewer, but larger, groups would either limit the possibilities of choosing the size as a lot of classes would have the same importance rating, or produce a need to internally rank the classes within the groups. However, it would also make the classification a lot easier. If for example a 3-point scale with values of low, medium and high was used, then predicting in which group a class actually belongs would not be as difficult of a task for the machine learner as when 10 different values are possible. This would of course improve the results overall, but as previously stated, it wouldn't allow to select very specific sizes of the condensed diagrams.

Furthermore, one more drawback of using a 10-point scale is that there are more differences in opinions among different employees. A class that is very unimportant would probably be rated as low on the 3-point scale by most individuals. However, using a 10-point scale, some people might rate the class as having the importance of 1, while others could rate it as 2, or maybe even higher.

Although both the precision and recall varied a lot between the different class importance values, it remained high for the importance value of 1. Thus, a lot of classes that actually had the importance of 1 were also predicted to be of the importance 1. Additionally, it can be seen in the confusion matrices that not very many important classes were predicted to have the importance 1. Therefore, the machine learner did very well in finding the least important classes.

As table 5.14 shows, the metrics prev_defects, defects, prev_versions and prev_prev_versions only managed to reach a score of 0. Furthermore, it was not possible to perform the cross validation nor to use the holdout method using the error metrics group which only included prev_defects and defects. It was suspected that there was something wrong with the retrieving of the metric, however, it was confirmed by one of the employees that many classes actually were very new and thus these metrics had yet not gotten higher values. Thus, the metrics were never removed from the analysis. Additionally, if the analysis would have been performed on an older system, perhaps these metrics would have yielded a higher result.

Furthermore, as table 5.14 shows, the groups of the highest scoring metrics varied a lot. While the two highest rated metrics both were design metrics, the subsequent two metrics belong within the complexity group while the next two are located in the change history group. Thus, it is not possible to say that all metrics in one group are better as predictors than all metrics in another group.

RQ2: Which metrics of a software class can advantageously be used as predictors in a machine learning algorithm to determine the importance of the class?
Design metrics, complexity metrics and Change history metrics work well as predictors. More specifically, the metrics IC_Par, EC_Par, token_count, cyclomatic_complexity, changed and added have the best information gain ratio out of all the tested metrics.

6.3 Performance of condensed diagrams

The ratings seen in table 5.15 show that only the condensed diagrams for diagram 3 performed badly. This was due to the fact that diagram 3 was a lot smaller than the other diagrams that were processed. If all diagrams were of the same large size, the results would probably have been both better and more consistent. However, it is yet left to be determined when a diagram is too large and needs to be condensed and how much it should be condensed, i.e. how many classes that should be removed from the diagram.

Feedback that was given included that in some cases, very important classes had been removed. This is definitely due to a bad prediction of the machine learner. This misbehaviour could possibly be avoided by further extending the used set of metrics. Additionally, by using a ground truth that is larger or more optimised for different stakeholders, this error could maybe be prevented.

One more flaw in the diagrams that was found was the lack of relations between classes in some cases. However, this was not due to bad predictions by the machine learner, but rather due to the quality of the reverse engineering process. The reverse engineered diagrams that were used had already a low amount of relations, and if these diagrams had been better, then the perceived quality of the condensed diagrams would have been higher as well.

Nevertheless, in all diagrams except diagram 3, the condensed diagrams were deemed as good or better than the reversed engineered one.

RQ3: Are class diagrams which are condensed through machine learning useful within industry?
Yes, overall the condensed diagrams performed better than reverse engineered diagrams with all classes in them.

6.4 Threats to validity

This section lists the possible threats to validity related to the work that was performed. The possible validity threats are categorised according to Wohlin et. al. [22].

6.4.1 Conclusion validity

The validation is due to the threat of *low statistical power*. As only five employees were part of the validation, the average rating of the condensed diagrams is by itself not very rigid. However, this threat was handled by gathering more qualitative feedback that explained the flaws and the benefits of the diagrams.

Furthermore, *reliability of measures* may affect the interviews as there of course could be misunderstandings. However the questions were designed to be very clear and easily understood, and further explanations were given during the interviews whenever a misunderstanding occurred.

Additionally, *reliability of measures* is relevant within the analysis of metrics. This is as no histograms were produced for the training and test set when the holdout method was used. Having a histogram showing the distribution for each of these sets could be useful for understanding the performance of the metrics. However, a histogram for the entire dataset was given to better understand the data.

6.4.2 Internal validity

As mentioned earlier, the lacking of relations between classes was seen as a problem with the condensed diagrams. However, this was not due to the machine learner itself but due to the quality of the reverse engineered diagrams. Thus, *instrumentation* affects the internal validity here. However, this threat was handled by reverse engineering diagrams through Enterprise Architect which is a widely used tool for the development process [23]. By using a state of the practice tool, the reverse engineered diagrams should be considered favourable compared to a not as extensively used tool.

Additionally, the ground truth was used as an instrument in the validation of the condensed diagrams and the analysis of metrics. However, this ground truth was created by a mix of design architects and developers. Thus, the ground truth is based on the opinions of different types of stakeholders. While it still managed to perform well, having different ground truths where each would be made by only one type of stakeholder could possibly improve the results further and make flaws and benefits of the method more obvious.

Furthermore, *selection* is a threat to the validity. While the selection aimed at being objective and sought out to include people with varying roles and using only people from different teams, the threat of selection could still affect the work. It could be that the selected individuals were among the better performing and more enthusiastic in their teams, or that the volunteers are more motivated than the average employee.

6.4.3 Construct validity

Mono-method bias could affect the construct validity. While several methods were used in the analysis of metrics, the gathering of opinions was done only through the use of interviews. It was considered to also use observation to further determine the information needs of the employees, however due to time limitation, this was never performed. Although only interviews were used to gather the opinons, background questions were sent to the interviewees to get more knowledge regarding their background and further strengthen their answers in the interviews.

6.4.4 External validity

Interaction of selection and treatment is a small threat to validity. This is because all the selected employees work within Ericsson. Thus, it is not certain that the same results would have been reached if another company had been targeted for the work. While working procedures and use of technologies may vary between companies, it is still reasonable to believe that the findings from Ericsson are valid in the settings of other companies as well.

6.5 Future work

There are two clear ways in which this work could be built further upon. First of all, as figure 5.7 shows, neither Change impact nor functionality is related to any group of metrics. While it would be difficult to find a good way to automatically determine what functionality a class is part of, change impact may not be as difficult. Thus, metrics related to the change impact could be tried in a future experiment to see if these improve the accuracy of the predictions.

Secondly, the list of codes found in the interviews could be extended and thus finding more possible groups of metrics. Additionally, more metrics could be used for the already existing codes. Also here, these new metrics could be used in a future experiment to improve the predictions.

Furthermore, the tool that was developed throughout the process can itself be used directly within industry in order to identify important classes within the system or even to automatically create condensed diagrams and produce a visualisation over the software. However, it would be beneficial for any company to try different strengths of condensation and not only use 60% or 80% classes remaining in the condensed diagrams as was tried within this thesis.

Additionally, the tool could also be built further upon to increase functionality or improve usability. As the tool in some cases would predict very important classes as not important at all, one way of adding functionality could be to allow users to manually set the importance of wrongly predicted classes and even include these in the ground truth. Additionally, building a more robust ground truth could improve the results of the tool and thus making the tool a greater asset within industry.

Chapter 7

Conclusion

The method of using machine learning to determine the importance of a software class in order to condense class diagrams to only contain the most important classes was very limited in terms of the metrics that had previously been used as predictors. To extend the set of metrics, several interviews were performed with different stakeholders to determine what they consider as important for a software class. Using the knowledge of what they found important, a tool was then built which is capable of condensing class diagrams. To see how well each metric that the tool uses performed, an analysis was carried out. This analysis used both the holdout method and cross validation to get an understanding of how correct the predictions of the machine learner was. Additionally, the information gain ratio of each metric used was calculated. Furthermore, the end product of the condensation, the condensed diagrams, were validated by comparing the flaws and benefits in relation to reverse engineered diagrams where all classes still remained.

The results showed that there were several aspects of software classes that were considered important for the various stakeholders. While some of these aspects were related to the design metrics that were previously used as predictors, most of them were not and could instead be linked to new metrics which had not been tried before. Furthermore, when these metrics were analysed, it was determined that out of the top performing metrics, there was a good representation of metrics from different groups. More specifically, some metrics related to the complexity of a class and to the change history scored very highly in addition to the existing design metrics.

The validation of the condensed diagrams was very positive. However, it also showed that there were some flaws with the diagrams. In some cases, important classes had received a low prediction and were removed from the diagram. In other cases, the employee indicated that more information, such as a description of a relation or a brief explanatory text would be needed to fully understand the contents of the diagram. Nevertheless, the diagrams overall received a good rating when comparing to the original reverse engineered diagrams. In only one out of five diagrams, the condensed diagram was deemed to be worse than the reverse engineered diagram.

References

[1] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice.* Addison-Wesley, 2013.

[2] P. Tonella, "Reverse engineering of object oriented code," *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*

[3] F. Thung, D. Lo, M. H. Osman, and M. R. Chaudron, "Condensing class diagrams by analyzing design and network metrics using optimistic classification," in *Proceedings of the 22nd International Conference on Program Comprehension*, pp. 110–121, ACM, 2014.

[4] M. H. Osman, M. R. Chaudron, and P. v. d. Putten, "An analysis of machine learning algorithms for condensing reverse engineered class diagrams," *2013 IEEE International Conference on Software Maintenance*, 2013.

[5] M. H. Osman, M. R. Chaudron, P. van der Putten, and T. Ho-Quang, "Condensing reverse engineered class diagrams through class name based abstraction," *2014 4th World Congress on Information and Communication Technologies (WICT 2014)*, 2014.

[6] G. Booch, *The unified modeling language user guide.* Pearson Education India, 2005.

[7] ISO/IEC 19501:2005, "Information technology - open distributed processing - unified modeling language (UML) version 1.4.2," 2005.

[8] R. S. Sutton, S. D. Whitehead, *et al.*, "Online learning with random representations," in *Proceedings of the Tenth International Conference on Machine Learning*, pp. 314–321, 1993.

[9] S. Kotsiantis, "Supervised machine learning: A review of classification techniques," *Informatica*, vol. 31, no. 3, 2007.

[10] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[11] Ericsson, "Company Facts - Ericsson." `http://www.ericsson.com/thecompany/company_facts`, 2015. [Online; accessed 18-February-2016].

[12] IBM, "IBM - Rational Rhapsody family - United States - Rational Rhapsody." `http://www-03.ibm.com/software/products/en/ratirhapfami`, 2016. [Online; accessed 18-February-2016].

[13] D. Steidl, B. Hummel, and E. Juergens, "Using network analysis for recommendation of central software classes," in *2012 19th Working Conference on Reverse Engineering*, pp. 93–102, IEEE, 2012.

[14] I. Sora and D. Todinca, "Using fuzzy rules for identifying key classes in software systems," in *Applied Computational Intelligence and Informatics (SACI), 2016 IEEE 11th International Symposium on*, pp. 317–322, IEEE, 2016.

[15] B. Bellay and H. Gall, "A comparison of four reverse engineering tools," *Proceedings of the Fourth Working Conference on Reverse Engineering.*

[16] M. Hammad, M. L. Collard, and J. I. Maletic, "Measuring class importance in the context of design evolution," *2010 IEEE 18th International Conference on Program Comprehension*, 2010.

[17] D. Sun and K. Wong, "On evaluating the layout of uml class diagrams for program comprehension," in *13th International Workshop on Program Comprehension (IWPC'05)*, pp. 317–326, IEEE, 2005.

[18] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2008.

[19] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *Software Engineering, IEEE Transactions on*, vol. 25, no. 4, pp. 557–572, 1999.

[20] S. E. Hove and B. Anda, “Experiences from conducting semi-structured interviews in empirical software engineering research,” in *Software metrics, 2005. 11th ieee international symposium*, pp. 10–pp, IEEE, 2005.

[21] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques.* Elsevier, 2011.

[22] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering.* Springer Science & Business Media, 2012.

[23] L. Khaled, “A comparison between uml tools,” in *Environmental and Computer Science, 2009. ICECS’09. Second International Conference on*, pp. 111–114, IEEE, 2009.

Appendices

Appendix A

Interview Guide

- Role
  - Work tasks?
- Get to know system.
  - What system are you working with?
  - Size of system?
  - Are you working on high level architecture? Or on a lower level with details?
- How do you make architecture decision?
  - Can you please explain the process from a requirement from a stakeholder to decision that affects the architecture
  - When there is a change of requirements so that you need to build new software or change existing software, how do you start working with the change?
- What diagrams do you regularly use? (Class diagrams, package diagrams,)
  - When in the process do you use the diagrams?
  - What information do you want to find when you read class diagrams?

- What makes a class important? And why?
  - What makes a class more important than another?
  - Does the number of dependencies make a class more important?
    - Type of dependency?
  - Does the number of attributes make a class more important?
  - Does the number of contributors make a class more important?
  - Does the number of bugs make a class more important?
  - Would class with perhaps a high cyclomatic complexity or a large number of lines of code be more important for you?
  - Does the fact that a class is changed by a lot of people make it more important to you?
  - Could you think of any more things that makes a class more important than another?
  - Which classes are the least important ones? Why?
- When discussing with other stakeholders regarding needed changes, what kind of information do you share/receive?
  - Do you use diagrams to support the communication?
  - What do you focus on when using the diagrams?
- Would a diagram where the least important classes have been removed be useful for you?