

# Classification of role stereotypes for classes in UML class diagrams using machine learning

Master's thesis in Software Engineering

Jobaer Ahmed  
Maoyi Huang



MASTER'S THESIS 2020

# Classification of role stereotypes for classes in UML class diagrams using machine learning

Jobaer Ahmed  
Maoyi Huang



Department of Computer Science and Engineering  
CHALMERS UNIVERSITY OF TECHNOLOGY  
UNIVERSITY OF GOTHENBURG  
Gothenburg, Sweden 2020

Classification of role stereotypes for classes in UML class diagrams using machine learning

Jobaer Ahmed

Maoyi Huang

© Jobaer Ahmed, Maoyi Huang, 2020

Supervisor: Michel R. V. Chaudron, Department of Computer Science and Engineering

Examiner: Riccardo Scandariato, Department of Computer Science and Engineering

Master's Thesis 2020

Department of Computer Science and Engineering

Chalmers University of Technology and University of Gothenburg

SE-412 96 Gothenburg

Telephone +46 31 772 1000



# Classification of role stereotypes for classes in UML class diagrams using machine learning

Jobaer Ahmed

Maoyi Huang

Department of Computer Science and Engineering

Chalmers University of Technology and University of Gothenburg

## Abstract

Software development process is becoming inherently complex in recent decades. To reduce the complexity in the development process developers, software practitioners are constantly looking for newer approach. One approach can be understanding the software design for instance, the UML models earlier in the software development process. For analyzing UML models, one could use knowledge about role-stereotypes. Knowledge about role stereotypes can help during software quality assessment, for summarizing software and thereby to ease the understanding of software designs. This study presents a machine learning-based approach for classifying the role-stereotype of classes in UML class diagrams. We have established a ground truth by manually labelling 391+ classes from 15 open source projects (using various programming languages). We analyze the performance of the machine learning approach with the manually established ground truth. Besides, we show a comparison between our approach and another machine learning approach from an earlier case study which is based on source code. Furthermore, we compare different machine learning (ML) algorithms to find out the best ML algorithm for classifying our dataset. Another noteworthy contribution of this study is an analysis of which features are most relevant for classifying classes into role stereotype and which features generate the best classification performance. According to our findings, the J48 classifier performs best when classifying the raw dataset and the Random Forest classifier performs best on a more balanced dataset which has been obtained by applying SMOTE oversampling. By using our classifier software developers can analyze patterns in their software design at the early stage of software development process.

Keywords: role-stereotypes, machine learning algorithm, classification, data analysis, data mining, UML class diagram, software design, software engineering.



# Acknowledgements

We would like to express our gratitude and thanks to Michel R. V. Chaudron for being our supervisor from the university and providing us guidance, support and direction throughout the thesis study.

We wish to extend our gratitude to Felix Dobslaw, Truong Ho-Quang, Rodi Jolal, Arif Nurwidyantoro and Bassem Hussein for helping us with their valuable suggestions, for participating in the meetings and for giving us feedback in person or through emails.

We would like to thank our examiner Riccardo Scandariato for his invaluable feedback and support.

Finally, we would like to thank everyone who gave us their valuable time, suggestions, feedback and supported us during our case study.

Jobaer Ahmed and Maoyi Huang, Gothenburg, April 2020



# Acronyms

**CT** Controller.

**CO** Coordinator.

**CRI** Class Role Identifier.

**FP** False Positive.

**IH** Information Holder.

**IF** Interfacer.

**MCC** Matthews correlation coefficient.

**ML** Machine Learning.

**RF** Random Forest.

**ST** Structurer.

**SP** Service Provider.

**SrcRI** Source-code Role Identifier.

**SMOTE** Synthetic Minority Over-sampling Technique.

**TP** True Positive.

**URI** UML Role Identifier.

**UML** Unified Modeling Language.

**WEKA** Waikato Environment for Knowledge Analysis.

---

# Contents

List of Figures	xiii
List of Tables	xv
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Statement of the Problem . . . . .	3
1.3 Purpose of the Study . . . . .	3
1.4 Research Questions . . . . .	4
<b>2 Background</b>	<b>5</b>
2.1 Software Design . . . . .	5
2.2 UML Class Diagrams . . . . .	5
2.3 Role Stereotypes . . . . .	7
2.3.1 Source Code Level . . . . .	8
2.3.2 UML Model Level . . . . .	8
2.4 Tools . . . . .	8
<b>3 Methodology</b>	<b>11</b>
3.1 Experiment Setup . . . . .	12
3.1.1 Lindholmen Database . . . . .	12
3.1.2 SDMetrics . . . . .	12
3.1.3 Weka Machine Learning tool . . . . .	12
3.2 Approach . . . . .	12
3.2.1 Data selection Criteria . . . . .	13
3.2.2 Data Collection . . . . .	13
3.2.3 Feature Extraction . . . . .	14
3.2.4 Define Role Stereotype Criteria . . . . .	15
3.2.4.1 Criteria regarding characteristics of classes . . . . .	15
3.2.4.2 Other Criteria . . . . .	16
3.2.5 Manual Labeling and Consolidation . . . . .	17
3.2.5.1 Manual Labeling and Refining . . . . .	17
3.2.5.2 Independent Evaluation . . . . .	17
3.2.5.3 Ground truth - Manual Labeling of Stereotypes . . . . .	19
3.2.5.4 Distribution of Occurrence of absolute numbers of role stereotypes . . . . .	21

3.2.5.5	Distribution of relative occurrence of role stereotypes	23
3.2.6	Experiment with Machine Learning Algorithm . . . . .	24
3.2.7	Analyze Classification Experiments . . . . .	26
<b>4</b>	<b>Results</b>	<b>27</b>
4.1	Evaluation of the Machine Learning Classifiers . . . . .	27
4.1.1	Multi-class classification for All <i>Confidence Level</i> . . . . .	27
4.1.2	Multi-class classification for High <i>Confidence Cases</i> . . . . .	30
4.2	Comparing Performance of different ML algorithms . . . . .	32
4.2.1	Deeper analysis of best performing classifiers . . . . .	33
4.3	Ranking of Relevant Features for predicting role-stereotypes . . . . .	37
4.4	Discussion . . . . .	39
4.4.1	Insights from the Ground truth - Manual labeling . . . . .	39
4.4.2	Comparison between role-stereotypes in UML Class diagrams vs in Source Code . . . . .	40
4.4.2.1	Manual Labeling perspective . . . . .	40
4.4.2.2	Machine Learning Perspective . . . . .	42
4.5	Threats to validity . . . . .	43
<b>5</b>	<b>Conclusion &amp; Future Directions</b>	<b>45</b>
5.1	Conclusions . . . . .	45
5.1.1	Topological features . . . . .	47
5.1.2	Features based on class-names . . . . .	47
5.2	Future Directions . . . . .	48
	<b>Bibliography</b>	<b>49</b>
<b>A</b>	<b>Appendix A</b>	<b>53</b>
<b>B</b>	<b>Appendix B</b>	<b>57</b>
<b>C</b>	<b>Appendix C</b>	<b>77</b>
<b>D</b>	<b>Appendix D</b>	<b>79</b>
<b>E</b>	<b>Appendix E</b>	<b>83</b>
<b>F</b>	<b>Appendix F</b>	<b>87</b>



# List of Figures

2.1	<i>Example of UML Class Diagram</i> . . . . .	6
2.2	<i>Role Stereotypes</i> . . . . .	7
3.1	<i>Methodology</i> . . . . .	11
3.2	<i>An example of Our Manual Labeling</i> . . . . .	17
3.3	<i>Color codes for role stereotypes</i> . . . . .	18
3.4	<i>Absolute of role stereotypes for each project</i> . . . . .	20
3.5	<i>Total number of stereotypes</i> . . . . .	20
3.6	<i>Total number of Classes for each project</i> . . . . .	21
3.7	<i>Histogram for absolute values</i> . . . . .	22
3.8	<i>Percentage of role stereotypes for each project</i> . . . . .	22
3.9	<i>histogram of the percentage of role stereotypes in projects</i> . . . . .	24
4.1	<i>Illustration of Classification Results for All Confidence Level</i> . . . . .	29
4.2	<i>Illustration of Classification Results for High Confidence Level</i> . . . . .	32
4.3	<i>Comparison Among Different Machine Learning Algorithms Performance</i> . . . . .	33
4.4	<i>Comparison between URI and SrcRI classifiers</i> . . . . .	42
B.1	<i>Bitys UML class Diagram</i> . . . . .	57
B.2	<i>JGAP UML class Diagram</i> . . . . .	58
B.3	<i>Pizza Delivery System UML class Diagram</i> . . . . .	58
B.4	<i>GreenHouseXmlParser UML class Diagram (part 1)</i> . . . . .	59
B.5	<i>GreenHouseXmlParser UML class Diagram (part 2)</i> . . . . .	59
B.6	<i>GreenHouseXmlParser UML class Diagram (part 3)</i> . . . . .	60
B.7	<i>JPMC UML class Diagram (part 1)</i> . . . . .	60
B.8	<i>JPMC UML class Diagram (part 2)</i> . . . . .	61
B.9	<i>JPMC UML class Diagram (part 3)</i> . . . . .	62
B.10	<i>JPMC UML class Diagram (part 4)</i> . . . . .	62
B.11	<i>Neuroph UML class Diagram (part 1)</i> . . . . .	63
B.12	<i>Neuroph UML class Diagram (part 2)</i> . . . . .	64
B.13	<i>Neuroph UML class Diagram (part 3)</i> . . . . .	65
B.14	<i>Xuml UML class Diagram (part 1)</i> . . . . .	66
B.15	<i>Xuml UML class Diagram (part 2)</i> . . . . .	67
B.16	<i>MarsSimulation UML class Diagram (part 1)</i> . . . . .	68
B.17	<i>MarsSimulation UML class Diagram (part 2)</i> . . . . .	69

B.18	<i>ACSUFRO UML class Diagram</i>	70
B.19	<i>ObjectCourseEnd UML class Diagram</i>	70
B.20	<i>BioclipseBrunn UML class Diagram</i>	71
B.21	<i>Java_client UML class Diagram</i>	72
B.22	<i>SE_project UML class Diagram</i>	73
B.23	<i>Talon UML class Diagram</i>	74
B.24	<i>Wro4j UML class Diagram</i>	75
C.1	<i>Accuracy of J48 classifier on different dataset</i>	77
C.2	<i>Accuracy of Random Forest classifier on different dataset</i>	77
C.3	<i>Accuracy of OneR classifier on different dataset</i>	78
C.4	<i>Accuracy of ZeroR classifier on different dataset</i>	78
D.1	<i>K9 Project Sequence Diagram 1</i>	79
D.2	<i>K9 Project Sequence Diagram 2</i>	80
D.3	<i>K9 Project Sequence Diagram 3</i>	81
E.1	<i>Structurer</i>	83
E.2	<i>Coordinator</i>	84
E.3	<i>Controller</i>	84
E.4	<i>ServiceProvider</i>	85
E.5	<i>InformationHolder</i>	85
E.6	<i>Interfacer</i>	86
F.1	<i>Distribution of role stereotypes in ACSUFRO and Bitys</i>	87
F.2	<i>Distribution of role stereotypes in Talon</i>	88
F.3	<i>Distribution of role stereotypes in MarsSimulation and Neuroph</i>	88
F.4	<i>Distribution of role stereotypes in xUML and BioClipsebrunn</i>	88
F.5	<i>Distribution of role stereotypes in XMLParser and Wroj4j</i>	89
F.6	<i>Distribution of role stereotypes in JGAP and Java Client</i>	89
F.7	<i>Distribution of role stereotypes in Pizza Delivery System</i>	89
F.8	<i>Distribution of role stereotypes in ObjectCourseEnd and JPMC</i>	90
F.9	<i>Distribution of role stereotypes in SE Project</i>	90

# List of Tables

2.1	Role stereotypes distribution in source code . . . . .	8
3.1	List of Projects with UML Class Diagram . . . . .	13
3.2	Absolute table for role stereotypes . . . . .	19
3.3	Percentage table for role stereotypes . . . . .	23
4.1	Classification Results for All Confidence Level . . . . .	28
4.2	Confusion Matrix for Random Forest classifier for the imbalanced dataset (All confidence level) . . . . .	28
4.3	Confusion Matrix for J48 classifier for the imbalanced dataset (All confidence level) . . . . .	28
4.4	Classification Results for High Confidence Level . . . . .	30
4.5	Confusion Matrix for RF classifier for the imbalanced dataset (High confidence level) . . . . .	31
4.6	Confusion Matrix for J48 classifier for the imbalanced dataset (High confidence level) . . . . .	31
4.7	Comparisons among different classifiers accuracy . . . . .	32
4.8	Accuracy table for J48 Classifier - All Confidence Level . . . . .	34
4.9	Accuracy table for J48 Classifier - High Confidence Level . . . . .	34
4.10	Accuracy table for J48 Classifier - All Confidence Level (SMOTE) . . . . .	35
4.11	Accuracy table for J48 Classifier - High Confidence Level (SMOTE) . . . . .	35
4.12	Accuracy table for Random Forest Classifier - All Confidence Level . . . . .	35
4.13	Accuracy table for Random Forest Classifier - High Confidence Level . . . . .	36
4.14	Accuracy table for Random Forest (RF) Classifier - All Confidence Level (SMOTE) . . . . .	36
4.15	Accuracy table for Random Forest Classifier - High Confidence Level (SMOTE) . . . . .	36
4.16	Comparison between Dataset with Regular data and Dataset with SMOTE . . . . .	38
4.17	Comparison between URI and SrcRI classifiers . . . . .	41
4.18	Confusion Matrix for RF classifier for the imbalanced dataset (SrcRI) . . . . .	43
4.19	Confusion Matrix for RF classifier for the imbalanced dataset (URI) . . . . .	43
A.1	List of Projects . . . . .	54
A.2	Repository Link of Projects from Table A.1 . . . . .	55
A.3	Repository Link of Projects from Table A.1 . . . . .	56



# 1

## Introduction

### 1.1 Background

In a real world, the occupation can help to define a person. Similarly characteristics of a software class offer the reader a chance to know their classes better. There are six types of characteristics used to generalize the software classes, as denoted by Wirfs-Brock [1]: Information Holder (IH), Interfacer (IF), Controller (CT), Structurer (ST), Service Provider (SP) and Coordinator (CO). They are also called role stereotypes [1] according to her work.

Each of the Role stereotypes depicts a type of software class that only serves one certain type of functionality. For example, IH tends to have more attributes than operations since its main purpose is to contain and serve information. However, as clearly as the boundaries for each of the role stereotypes are defined, there is a lack of way to systematically and practically distinguish the roles stereotypes between software classes. Especially on the design level, there haven't been a single published standard that can directly label the characteristics of the class.

There are many ways to classify the role stereotypes on code level, such as the work from M.R.V.Chaudron & Ho Quang Truong [3]. They listed a couple of machine learning algorithms and scientifically evaluated each of them and finally made a decision on which algorithm is best performed regarding the classification of the roles in source code.

Likewise, in this research study, we do the same work but on a different scope. In our case, our main focus is on the design level which is within the UML model. We strive to find out if there is a feasible and efficient machine learning algorithm to classify the role stereotypes for software class within the UML model. In our study, similar aspects as the classification in source code are considered: coupling level, private/public modifier of the class, Dependencies and so on.

There are also aspects that are unique for UML models considered, for example, the name of the class. Arguably the class name is the most obvious and informative indicator to illustrate the purpose of the software class. And if the person who designs the software model follows the suggested pattern, we will have very easy and categorizable role stereotypes. For example, a Controller class normally does make a decision during the entire software functionality cycle. So its name tends

to be a operator-type such as "manager", "controller" or simply something with a "control" in the end. Thus, it is feasible to detect a "CT" type of class. However, in most of the times the class name is poorly designed due to either lack of software design practices or the long evolving time for the software design model to get updated.

The structure of the class is supposed to be easy-to-understand for the user. If the time that it takes for comprehending the software class can be reduced in a way, such as categorizing the software classes into a certain pattern, it may save tremendous effort for the engineers to perform tasks during their development stage. It can come in handy in certain circumstances. For example when they are receiving changing requirements, adding new features and maybe adopting new technologies. The concept of Role stereotype denoted by Rebecca conforms to this idea [1] [3]. She suggests to assign a "role" to the software class so the class can be systematically distinguished. It can help in various tasks such as software comprehension, software refactoring and quality assurance etc. These tasks all require extensive knowledge about the software architecture. So they can greatly help the engineer to speed up the process of doing their tasks. They can achieve to do the same amount of work but with less effort. In the 21<sup>st</sup> century world, needless to say, saving time and effort equals to saving money [11][12].

In this paper, we will present a research study that classifies the role stereotypes for UML models. The target of the UML models will be in range of the typical UML diagrams such as class diagram, sequence diagram and domain models, depending on their characteristics and the revealed information from the diagram. In our case, we mainly focus on the class diagram since its provided features to describe the software, i.e. classes, attributes, operations/methods and the relations among them are suitable candidates in order to classify the software classes into a certain pattern or stereotype. So, in order to make accurate classification, we will take use of the provided features from the UML class diagram. We will define the ground truth of the features so we know which role stereotypes can be detected by using a certain composition of the features. For example, a interface stereotype is suggested to have high number of attributes and little to none operations, according to Wirf-Brock's work [1] and Chaudron & Truong's selection criteria for source code [3] [9]. In order to make sure the role stereotypes we identify at UML models are valid and justifiable, we will conduct a manual inspection joined by Prof. Michel Chaudron and Ph.D candidate Truong to determine viable picks. Then we run automated identification algorithms to use the defined ground truth to detect the stereotypes in a training data set. The training data is set to be the lindholmen data base [13] along with the Github software project repositories. The former will be the primary resource since it contains over thousands of software projects with its UML model defined. The Weka [14] [15] machine learning tool which specializes in data mining tasks is the one to use after, for the automated classification process. When the training data set is processed through we will compare and analyze the result and see if there is any amend to be made. If it is necessary the already defined ground truth will be changed and apply on the training data set again. When the ground truth is refined and ready, we will apply it to the testing data set. The result acquired from the

testing data set will be presented and analyzed in this study.

## 1.2 Statement of the Problem

According to the paragraphs before, we know that it is beneficial to apply role-stereotypes to characterize classes. However, there are not so many studies about finding the characteristics of the software in the design level. There are successful research studies that have categorized the characteristics of the software from its source code.

In software development process UML diagram or design of the software are more consistent than the source code. Source code can change in any stage of the software development cycle. So, finding role-stereotypes based on UML diagram is more secure.

For getting more coverage of the data and solving the data scarcity problem that we face while using the regular classification technique based on the source code.

New engineers faces problem with understanding a new system. As they have to look at the system with thousands lines of code without even knowing the systems behavior, dependencies. If we can define the systems behavior based on the stereotypes we will find from the UML diagram, we can reduce the hurdle of new engineers to comprehend the system. Besides, the company can invest less time and money for the new engineers.

## 1.3 Purpose of the Study

Chaudron et al. presented a ML-based classifier which classifies classes in Java onto their stereotype based on features extracted from source code [3]. Our case study followed another approach to build a ML-based classifier, where we extracted features from the design level which is the UML diagram of the software development process. The main goal of our research work is to use the extracted features from the UML diagram for characterizing different classes and labeling them based on the role-stereotypes.

Role stereotypes can help in various tasks in software development and maintenance such as program design, program comprehension, summarising [3], [4], quality assurance [33], and reverse engineering [34], [35]. This case study proposes an automated machine learning-based approach for classifying role-stereotypes of classes in the design level [2]. At first, we have selected 15 to 20 projects and collected its UML diagrams. Then, we have extracted the features from the UML diaram which would be used by the machine learning algorithms. Next, we defined the ground truth. Later we used those features for characterizing and labeling the role-stereotypes in all classes.

In this case study, we predicted the behaviour of the classes based on the role-stereotypes that we will find from the UML diagrams. Based on the results, we could discover the relation among different classes. We figured out if there is high

coupling and low cohesion or vice versa between 2 classes.

### 1.4 Research Questions

The main purpose of our research work is to establish a machine learning (ML) based classifier which will classify all classes which exists in the UML class diagram. In order to achieve this, we set up research questions as the follows:

- RQ 1. How can Machine Learning be used to build a useful classifier for role stereotypes of classes in UML class diagrams?
  - RQ 1.1. Which features are useful for identifying role stereotypes in UML class diagrams?
  - RQ 1.2. Which machine learning algorithm yields the best performance in classifying role stereotypes?
- RQ 2. How does the classifier of classes in UML diagrams compare to the existing classifier for classes in source code?

RQ1 is broken down in two parts: RQ 1.1 studies feature selection criteria to determine whether or not a feature is able to be used to classify stereotypes. RQ 1.2 studies which ML algorithm yields the best performance. The performance of the ML algorithm is evaluated against a ground truth of manually labelled classes. In addition to this, we studies whether the performance of our classifier is better or worse than the existing classifier for classes in source code [3]. This is captured by RQ2.



# 2

## Background

This study is carried out as a continuation of a previous study: "Improving the Automated Classification of Role-Stereotypes by Machine Learning" [3]. The previous case study serves as a path finder for the research, and it will also be a facilitation for any future works.

In the following sections, we will introduce some related works that inspired us to get our desired solutions for this research study.

### 2.1 Software Design

The idea of design is connected to the human characteristics, which is one of most distinctive one. These characteristics are the making and use of tool. Tools are artifacts, which can be used to create more artifacts. And, producing any form of artifact is an act that uses some element of design activity. Another human characteristic is communication. Converting the design into a product needs communication, to convey the idea to the development team so that they can develop the design into a product. The product can be a software or physical object. There are various artifacts, which are the results of various applications of the design process plays an influential role in our daily lives. For example: we ride in cars, trains, airplanes, we live in houses or flats etc. are the products that are outcome of the design process [18].

Similarly, in case of software system, design plays a vital role. Majority of the people will think, bigger system needs to be well designed and precisely tested. But good design is necessary for smaller systems as well. As the user needs efficiency, reliability irrespective of the size of the system. Although, there is a high exposure in the design process during the software development, good design practices are not followed rigorously. In general, the way people carry out the design process is not structured [18]. In the area of computer science and software engineering, software design is one of the main problem-solving technique besides the notion of theory and abstraction [19].

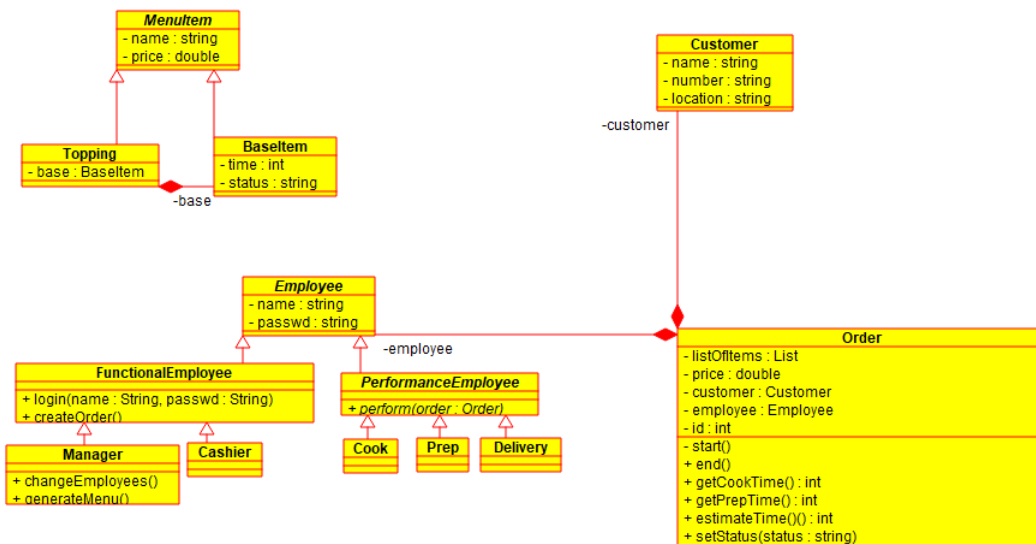
### 2.2 UML Class Diagrams

In traditional code-centric development, developers uses simple sketches for design ideas, often they don't even store the sketch for future use. It was sufficient at that

## 2. Background

period. In model driven approach the models are the primary source for developing an optimal software. A clear understanding of models are required to create a structured model. For instance, an UML model can be used to describe software design, pattern and processes [22]. An example of UML class diagram is illustrated in the figure 2.1

Chaudron et al.[21] discussed the gaps that are identified during effective UML modeling in his paper. Furthermore, he described the empirical evidence of the usefulness of UML modeling in software development. His research mainly focused on the costs and benefits, and on industrial practice. He mentioned, modeling for analysis and understanding or modeling as a sketch is loose style of modeling. Because, that way was followed for personal understanding and it can be done on a white board. By developing a structured UML diagram, a developer can reduce the cognitive complexity to manage huge details of the design in his mind. There



**Figure 2.1:** *Example of UML Class Diagram*

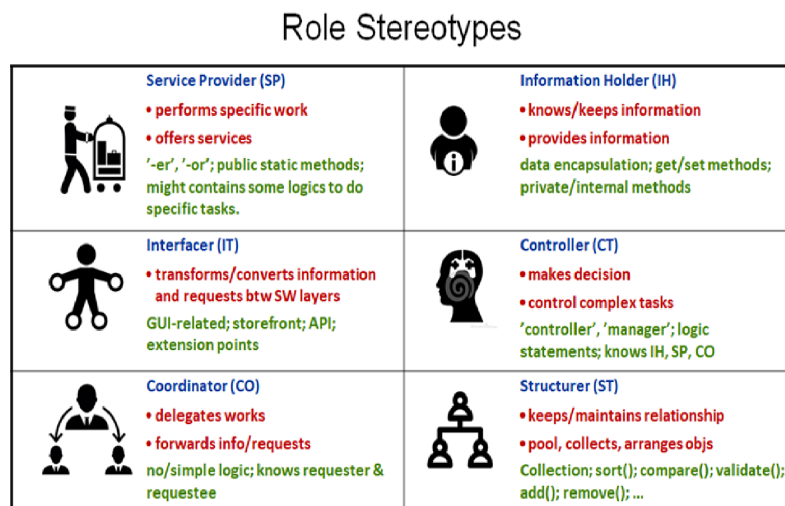
are many types of UML diagrams, among them UML class diagrams are a key resource for developing object oriented software system. Because they establish the ground for the future design and development. It can be said that, if the quality of the UML class diagram is significantly higher, then the software system will be of higher quality. In this modern era quality of the software is really important. Software quality should be maintained from the early stage of the development cycle [23].

The quality assurance methods are more effective in the initial stage of development, rather than applying them in the end of the development cycle. Detecting bugs and fixing them is more expensive during the last stage of the development life cycle. So it is better to put some more effort during the design phase of the software system, then it will be easier to avoid unwanted cost [23].

## 2.3 Role Stereotypes

As noted by Rebecca Wirfs-Brock's [1], assigning a role to a class characterizes initial candidate objects and communicate designer's ideas to whomever use the system later [1]. If a class is well-defined and distinguishable, it can even propagate some level of implementation details to the developers in an early stage. Also she noted that the role stereotype is not only for designing new objects, but also it can be used to dissect the design patterns of a software [1].

One of the commonly accepted ideology from Wirfs-Brock's classification rule is that there are 6 types of role stereotypes in software. Each of them serves a certain purpose and they complement each others' functionality. This idea is also supported by Prof. Michel Chaudron and Dr. Ho Quang Truong in their criteria of classification for the classes[3]. For more information about the stereotypes, it can be viewed in the graph here:



**Figure 2.2:** *Role Stereotypes*

As shown in figure 2.2, the role stereotypes are:

- Information Holder(IH): An object designed to know certain information and provide that information to other objects.
- Structurer(ST): An object that maintains relationships between objects and information about those relationships.
- Service Provider(SP): An object that perform specific works and offers services to others on demand.
- Controller(CT): An object designed to make decisions and control complex task.
- Coordinator(CO): An object that is not involved with making decisions, but in a way delegates work to other objects. E.g., SP, CT etc.
- Interfacer(IF): An object that transfers and changes information or requests between distinct part of the system.

### 2.3.1 Source Code Level

It can be beneficial to have extensive research studies about the identification on role stereotypes. So far most of the works are conducted on source code level, in this section we will introduce some related works that presented some intricate findings regarding the roles stereotypes on source code level.

In the study of Chaudron et al. (2019), Automated Classification of Class Role-Stereotypes via Machine Learning, they suggest that there could be interrelations between role stereotypes and there are patterns that can be found across software projects [3]. For example, if there is high percentage of service provide (SP) found in a software project comparing to the other role stereotypes, it indicates that this software project has a high chance of being a daily routine software program such as E-commerce platform or a workout program.

In their study they have collected an extensive number of projects which consist of 779 Java classes in total, and they have done manual labelling to label each class. After that, they ran several machine learning algorithms to testify the labels are given correctly. In the result, they discovered a distribution of the role-stereotypes across all the Java classes (for a handful of projects). We attach the distribution result in Table 2.1:

Role	IT	CO	CT	SP	IH	ST	Total
Dist. Abs	77	79	20	323	231	49	779
Dist. %	9.9%	10.1%	2.5%	41.5%	29.7%	6.3%	100%

**Table 2.1:** Role stereotypes distribution in source code

### 2.3.2 UML Model Level

As important as the stereotypes existed in source code level, the role stereotypes are equally vital on the design level. To some extent, the UML model requires more precise measurement and precision calculation of real-world geographical entities such as the Global Navigation Satellite System (GNSS) [31]. Michel and Truong [9] also notified that the UML class diagrams are used for designing and describing the architecture of the software, they are an essential tool for the engineers to understand the basic structure of the system. In terms of industrial and academic perspective, it is even more beneficial to grasp the characteristics of the UML model's stereotype [9]. According to Moha and Yann-Gaël's work, there are also a special type of stereotypes for UML models which is called the Anti-pattern [10]. The anti-pattern indicates to a bad design solution to a software. It can be detected by algorithms that can capture the certain number of features. We use the anti-patterns as a spare equipment to detect the UML stereotypes in this study.

## 2.4 Tools

Alhindawi et al. demonstrated in his paper that source code stereotypes reduces the overall effort of a developer to find the suitable methods for extracting features [6].

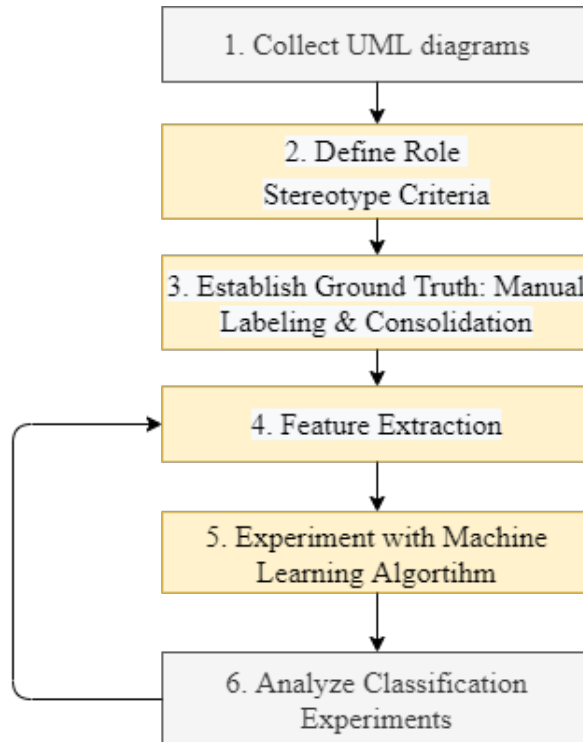
For identifying method stereotypes Dragan et al. have classified the different stereotypes using a taxonomy which occurs frequently [7]. Their proposed method is mainly based on the C++ programming language. A method can be labeled with one or more stereotypes. Although the proposed method was developed for mainly C++ projects, it can be useful for Java-based projects or other types of projects. In another case study, Moreno et al. presented a tool, which can automatically detect the source code stereotypes in java-based projects [8]. It works as an eclipse plugin and can classify any java project based on the discovered stereotypes from the methods and classes.



# 3

## Methodology

Figure 3.1 shows an overview of our research methodology. First we have collected projects with UML class diagram in .xml, .xmi and .uml file format. Besides, we have prioritized the projects we have used based on different factors. Next, we have input the selected projects in the SD Metrics tool to analyze and to extract the first batch of metrics or features we have used during our experiment with machine learning algorithms. We have established our ground truth by manually labeling approximately 400 classes from 15 projects. Here, stage 4, 5 and 6 are an iterative process and we went back to the former stages several time to refine our features. Then, in stage 5, we experiment our features with various machine learning algorithm and evaluate their performance. In the final stage, we have presented and analyzed our experiment results based on our classifier "UML Role Identifier (URI)".



**Figure 3.1:** *Methodology*

## 3.1 Experiment Setup

In this section, we will describe the tools and technologies used for this research study. Since the research is divided into several stages, there will be tools for certain stages and for some stages i.e. manual labelling and analyze classification result will have no specific tools which needs to be systematically operated in order to get the result.

### 3.1.1 Lindholmen Database

Lindholmen database is an open source database [17] which collects over 93000 UML models across more than 24000 github repositories. The purpose of establishing this database is to assist the academic and also industrial researchers to get access to the open source software projects which contains UML. From Industrial perspective, the use of UML class diagrams were studied extensively. On the other hand, there are not much information about the UML uses in Free/Open source software (FOSS) projects. The main goal of this dataset is to find out if the models which were used in the software projects are updated throughout the project's life cycle or not. For collecting the datasets, the researches have used a semi-automated approach to collect the UML, which are stored in images, .xmi and .uml format. They have scanned 10 percent of all Github projects (1.24 million). After gathering all the information they have analyzed the models and found that 12% of the models are duplicated. In conclusion, they have prepared a list of Github projects that include UML files. In our case study, we have used this list, to collect projects with .xmi files, which we have used for extracting features.

### 3.1.2 SDMetrics

The SDMetrics is the software that we have used to extract our features during the fourth step of our methodology 3.1. We have used this tool for getting an overview of all the projects, that we have used in this case study. With its built-in design rules and well-refined criteria in terms of deciding the design models, it can help us to check various aspects of our UML design for its completeness, consistency, correctness, design style issues such as dependency cycles, and more.

### 3.1.3 Weka Machine Learning tool

The tool is a collection of machine learning algorithms [14] [15] specifically designed for data mining tasks. The Weka contains algorithms for data preparation, classification, regression, clustering, association rules mining and also visualization. It is also an open source software issued by the GNU General Public License.

## 3.2 Approach

In this section, the first 5 steps from our methodology will be described in detail. The last step **Analyze Classification Experiments** will be described in the following



chapter, which is the result chapter.

### 3.2.1 Data selection Criteria

1. Projects uses UML class diagrams
2. Any software programming language (no restriction)
3. The UML design is represented in a format of XMI, XML (.uml file types)
4. Quality assurance
  - Diagram must be a Forward Design (not Reverse Engineered)
  - Must be able to read class-, method- and attribute-names (for verification)
  - No non-UML elements in the diagram
  - Number of classes  $> 8$  , with ‘some’ attributes & methods
  - Not poorly designed. e.g., one class relates to all
  - Not many ‘orphan’-classes (without relations)

### 3.2.2 Data Collection

Name of the Project	No. of classes in the UML
1. Neuroph	24
2. Mars-simulation	40
3. Wro4j	20
4. JGAP	18
5. Java_Client	57
6. JPMC	24
7. ACSUFRO	9
8. Bioclipse-brunn	42
9. Bitys	11
10. Green_House_XML_Parser	31
11. Object_Course_End	12
12. Pizza_Delivery_System	13
13. SE_Project	30
14. Talon	15
15. XUML	45

**Table 3.1:** List of Projects with UML Class Diagram

In this case study we have collected 30+ projects (Look at Table A.1) from Lindholmen dataset [17]. From those projects we have selected 15 of them which has .xmi or .xml files, which are shown in the table 3.1. Reasons for choosing projects with .xmi file.

- At first we have selected project with UML class diagram image file. In that case, we couldn’t find images in high resolution for some projects and our tool couldn’t analyze the images because of the low resolution.
- The scarcity of projects with UML class diagram image file.

Besides, we have made sure that the .uml, .xml and .xmi files are not extracted from the reverse engineered uml class diagram images.

In our initial step, we have considered all the classes from the 15 projects that we have selected for this case study. For establishing the **ground truth** for this case study, we have taken an iterative approach from step two to five (see figure: 3.1).

#### 3.2.3 Feature Extraction

The features or metrics that are used for classifying the classes will be listed here. There are two types of them:

- One type of metrics are derived from the SDMetrics (3.1.2)
- The other type of metrics are created manually, based on our findings.

SDMetrics (3.1.2) is a popular tool, which is used by software practitioners during feature extraction from UML models. We have extracted several metrics from SDMetrics, from those metrics we have selected the metrics which have values and removed others which doesn't have any values. So the selected metrics are as following:

1. Metrics derived from SDMetrics.
  - **NumAttr**: The number of attributes in the class
  - **NumOps**: The number of operations in the class
  - **NumPubOps**: The number of public operations in the class
  - **Setters**: The number of operations with a name starting with 'set'
  - **Getters**: The number of operations with a name starting with 'get', 'is' or 'has'
  - **Nesting**: The nesting level of class (for inner class)
  - **IFImpl**: The number of interfaces the class implements
  - **NOC**: The number of children in the class (UML generalization)
  - **NumDesc**: The number of descendants of the class (UML generalization)
  - **NumAnc**: The number of ancestors of the class
  - **DIT**: The depth of the class in the inheritance hierarchy
  - **CLD**: Class of leaf depth
  - **OpsInh**: The number of inherited operations
  - **AttrInh**: The number of inherited attributes
  - **Dep\_Out**: The number of the elements on which this class depends
  - **Dep\_In**: The number of the elements that depend on this class
  - **NumAssEl\_ssc**: The number of associated elements in the same scope of the class
  - **NumAssEl\_sb**: The number of associated elements in the same scope branch of the class
  - **NumAssEl\_nsb**: The number of associated elements not in the same scope of the class
  - **EC\_Attr**: The number of times the class is externally used as an attribute type

- **IC\_Attr**: The number of attributes in this class having another class or interface as their type
  - **EC\_Par**: The number of times the class is externally used as a parameter type
  - **IC\_Par**: The number of parameters in this class having another class or interface as their type
2. Newly added metrics.
    - **EndWithManager, Controller**: The number of Boolean flag for classes with a name ending with 'Manager', 'Controller' or 'Control'.
    - **HasType, Annotation, List**: The number of Boolean flag for classes with a name containing 'Type', 'Annotation', 'List' or 'Data'.
    - **EndWithFactory, Impl, Implementation**: The number of Boolean flag for classes with a name containing 'EndWithFactory', 'Impl' or 'Implementation'.
    - **isEntity**: The number of Boolean flag for classes which are entities. An entity can be a person, place, or object. For instance: Customer, Employee, Car etc.

### 3.2.4 Define Role Stereotype Criteria

At first we have decided to establish some initial set of criteria, which can be used during our manual labeling stage of the methodology 3.1. That means, we have set these criteria so that we can follow them while establishing our ground truth. We have refined these criteria of selection based on our final results. The initial idea was taken from the paper written by Wirfs-Brock [1].

In the following paragraph, the criteria is listed for each role stereotype which was mentioned by author Wirfs-Brock in her paper [1]. The selection criteria for role stereotypes can be divided into 2 categories:

1. Criteria regarding characteristics of classes.
2. Other Criteria.

#### 3.2.4.1 Criteria regarding characteristics of classes

These criteria focuses on the attributes, name of the class, functions and method names. Some of the criteria can be a bit similar to the work of Dragan, Moreno and Chaudron. But, in our case study we have considered them for labelling classes in UML class diagrams, where as they have used them for labelling classes in source code. They have particularly focused on the size, frequency and the magnitude while labelling the classes [7][8][3].

1. **Information Holder**: An object designed to know certain information and provide that information to other objects. Selection Criteria:
  - (a) If a class is with type ENUM (metrics TBD)
  - (b) If the class ends with DATA, GEO, CONFIG, CMD, REQ
  - (c) Class name may contain "-Type", "-Annotation", "-List"
  - (d) Class name may be an entity. e.g. "User"
  - (e) May contain getters/ setters.

- (f) May contain data/information/info in their class name.
  - (g) May be represented as enum class
  - (h) Can be an interface, if its methods are only setters and getters (in general: giving access to its attributes)
2. **Structurer:** An object that maintains relationships between objects and information about those relationships. Complex structures might pool, collect, and maintain groups of many objects; simpler structures maintain relationships between a few objects. Selection Criteria:
    - (a) It might have composition or aggregation relationship with its subclasses.
    - (b) Has method(s) to maintain relationships between objects
    - (c) Methods that manipulate the collection such as sort(), compare(), validate(), remove(), updates(), add(), etc.
    - (d) Methods that give access to a collection of objects such as get(index), next(), hasNext(), etc.
  3. **Service provider:** An object that performs specific works and offers services to others on demand. Selection Criteria:
    - (a) Class name may end with “-er” (eg. Provider) or “-or” (eg. Creator, Detector)
    - (b) Class name may end with “Impl”
    - (c) Class name may end with “Function”
    - (d) Class name may contain “Factory”
    - (e) Class name may contain "Listener" or "Exception"
    - (f) Class name may contain "Processor" or "Operator"
  4. **Controller:** An object designed to make decisions and control complex task. Selection Criteria:
    - (a) Class name may ended with “Controller” or “Manager”
    - (b) Have access to Information holders, coordinators or service provider
  5. **Coordinator:** An object that doesn’t make many decisions but, in a rote or mechanical way, delegate work or other objects. Selection Criteria:
    - (a) Class name may contain “Connection” or "Connector"
    - (b) Class name may contain "Binder" or "Event"
  6. **Interfacer:** An object that transforms info or requests between distinct part of the system. The edges of an application contain user-interfacer objects that interact with the user and external interfacer objects, which communicate with external systems. Interfacer also exist between subsystems. Selection Criteria:
    - (a) Class name may contain “Abstract”
    - (b) High values in NumOps, NumPubOps and NumDesc metrics.
    - (c) Class name may contain "<interface>" tag or label.
    - (d) Exception case: It might contain "Factory".

#### 3.2.4.2 Other Criteria

Labelling classes can become trickier when some classes represents dual or multiple roles. This concept is also mentioned by Wirfs-Brock [1] and Dragan [7]. These are the exception cases which were discussed during our meeting with experts. And, we have labelled the classes based on those discussions.

### 3.2.5 Manual Labeling and Consolidation

This section can be divided into two more subsections:

1. Manual labeling and refining the labeling based on the criteria of selection.
2. Independent evaluation of the labelled classes by the experts.

#### 3.2.5.1 Manual Labeling and Refining

We have followed an iterative approach while manual labeling the selected 15 projects (check Table 3.1). Each of the two authors labelled the classes of the projects individually. They have labelled the projects one at a time. After labeling each project, they had a discussion regarding the differences between their classification. Based on the discussion, they have refined the criteria of selection. An illustration of our manual labeling is shown in the figure 3.2.

In this figure 3.2, we have shown the initial labeling we did for ACSUFRO. After discussion, we have refined our labeling and added them in our final dataset. We went through this process for 4 times, until we have reached a point where the criteria is refined and can define our labeling adequately. For each project, we have spent 6 hours, that means for 15 projects we needed 90 hours time. Each of the author spent 45 hours and the overall manual labeling process took approximately 12 weeks. Finally, we have established a ground truth merging 391 classes from 15 projects.

7. Project Name: ACSUFRO (No. of classes: 9)			
Class Name	Role Stereotype	Agreed?	Comments
UML Model.Logical View.cl.ut fsm.acs.acg.core.DAOManager	Controller	yes	
UML Model.Logical View.cl.ut fsm.acs.acg.core.AlarmSystemManager	Controller	yes	
UML Model.Logical View.cl.ut fsm.acs.acg.core.AlarmsManager	Controller	yes	
UML Model.Logical View.cl.ut fsm.acs.acg.core.CategoriesManager	Controller	yes	
UML Model.Logical View.cl.ut fsm.acs.acg.core.SourcesManager	Controller	yes	
UML Model.Logical View.cl.ut fsm.acs.acg.core.ReductionsManager	Controller	yes	
UML Model.Logical View.com.cosylab.acs.laser.dao.ACSAlarmDAOImpl	Coordinator	no	service provider?
UML Model.Logical View.com.cosylab.acs.laser.dao.ACSCategoryDAOImpl	Coordinator	no	service provider?
UML Model.Logical View.com.cosylab.acs.laser.dao.ACSSourceDAOImpl	Coordinator	no	service provider?

**Figure 3.2:** *An example of Our Manual Labeling*

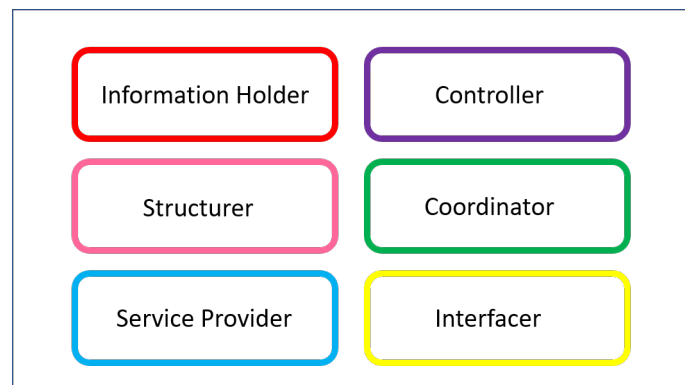
#### 3.2.5.2 Independent Evaluation

After the manual labeling process we have created colored UML class diagrams for the 15 projects. In figure 3.3, we have illustrated the color codes we have followed for each role stereotypes while labeling the classes in the UML class diagrams. We have provided these colored UML class diagrams and the corresponding excel sheet with labelled classes to our supervisor and one PhD student for evaluation. Both of them evaluated the resources separately. Later, we had several meetings for discussing the differences in our classification.

At this point, we have planned to use a label to show our confidence while labeling the classes. We have named this label as "Confidence Level". We have set a scale for this label, which is one to five, where one shows the lowest 'Confidence Level' and five shows the highest "Confidence Level".

When a class shows multiple roles, we have used this label to show our confidence while labeling the classes with role stereotype. If we have put a label five, that means, the class doesn't show dual role and everyone involved in the labeling agreed about it. When we have labeled the classes with 'Confidence Level' one to four, that means the classes are playing a dual or multiple roles, and we have disagreements.

We have made 2 datasets, one dataset has 391 labelled classes (All Confidence cases) and another dataset has 328 classes (High Confidence cases). We have put a label five confidence on a class, when we all have agreed on certain role stereotype for that class. For other classes we have put a label from one to four confidence level, based on how many have agreed or disagreed. The dataset with High confidence cases refers to the fact that we didn't have any disagreements when labeling the 328 classes on this dataset. If we remove the high confidence cases from the all confidence cases then we have 63 classes. We had disagreements during labeling these 63 classes, and we went for the popular choice while labeling them. That means, we have agreed for most of the labeling, and we didn't agree when a class is showing dual or multiple role. Based on our discussion the criteria for selection, labeling and the dataset was refined. We made sure all our data are consistent throughout the case study.



**Figure 3.3:** *Color codes for role stereotypes*

Some of the colored UML class diagrams are added in the Appendix B as a reference. We have created a repository where all the UML class diagrams and other resources from our case study will be found <sup>1</sup>. For now we have skipped adding all of them in the appendix, so that we don't over-flood the thesis report.

---

<sup>1</sup><https://github.com/hammer007/umlRoleIdentifier>

### 3.2.5.3 Ground truth - Manual Labeling of Stereotypes

In this section, we will show the raw data table which is called the "Absolute table for role stereotypes". Then we have the percentage table which takes the calculated data and shows it in percentages as a way to present the role stereotypes' distribution among projects. After that we generated a stack-columned chart with those data. For an overview, we have graphs for total number of project, total number of stereotypes. On top of that, we also have generated some histograms to show the data in an statistic view.

#### The absolute table & charts

In section 3.2.5, we conducted our manual labeling and consolidation step. We examined a set of software projects from Lindholmen Data Set [17] and finalized with 15 projects. When we completed the manual labeling process in iterations, we have recorded the number of instances of each role stereotype in every project. Finally, we have established a ground truth of 391 classes from those 15 projects. As a result, we have a table to show the numbers recorded as below:

Absolute of role stereotypes for each project							
Project List	IH	SP	IF	CT	ST	CD	Total
Java_Client	47	2	1	7	0	0	57
xUML	22	5	7	0	11	0	45
Bioclipse-brunn	20	8	10	0	0	4	42
Mars-simulation	32	0	0	4	4	0	40
GreenHouseXMLParser	6	15	1	9	0	0	31
SE_Project	1	16	7	0	5	1	30
Neuroph	3	15	0	0	6	0	24
JPMC	9	7	4	2	0	2	24
Wro4j	4	9	6	1	0	0	20
JGAP	7	7	3	0	0	1	18
Talon	13	2	0	0	0	0	15
Pizza_Deliver_System	0	7	3	1	2	0	13
ObjectCourseEnd	3	3	1	3	0	2	12
Bitys	0	7	11	0	0	0	4
ACSUFRO	0	0	0	6	0	3	9
<b>Total</b>	167	103	47	33	28	13	391

**Table 3.2:** Absolute table for role stereotypes

In the given table 3.2, we can see that there are many cases that it has a value 0 for the stereotype within the project and it happens quite often. There are extreme cases that one stereotype takes a significant number of instances when we compare it with the others for that single specific project, e.g., Java\_client with 47 IH identified. And last but not the least, the number of role stereotypes are not quite distributed evenly execept for project ObjectCourseEnd, with the value of 1-3 on each identified role stereotypes. For the reference, we added one more figure which shows the percentage of the absolute number below in figure 3.3.

### 3. Methodology

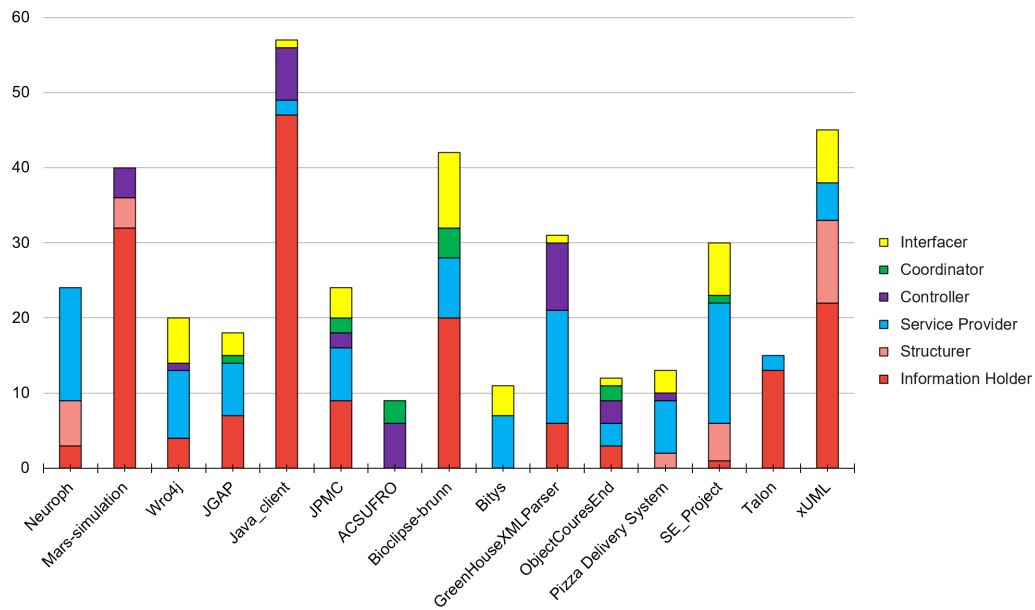


Figure 3.4: Absolute of role stereotypes for each project

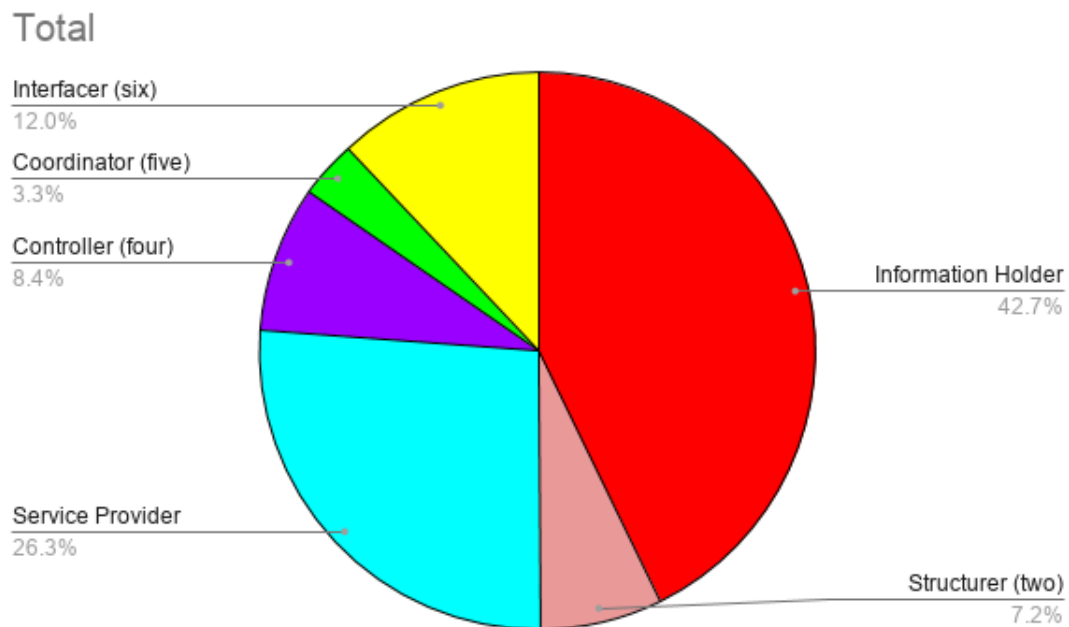
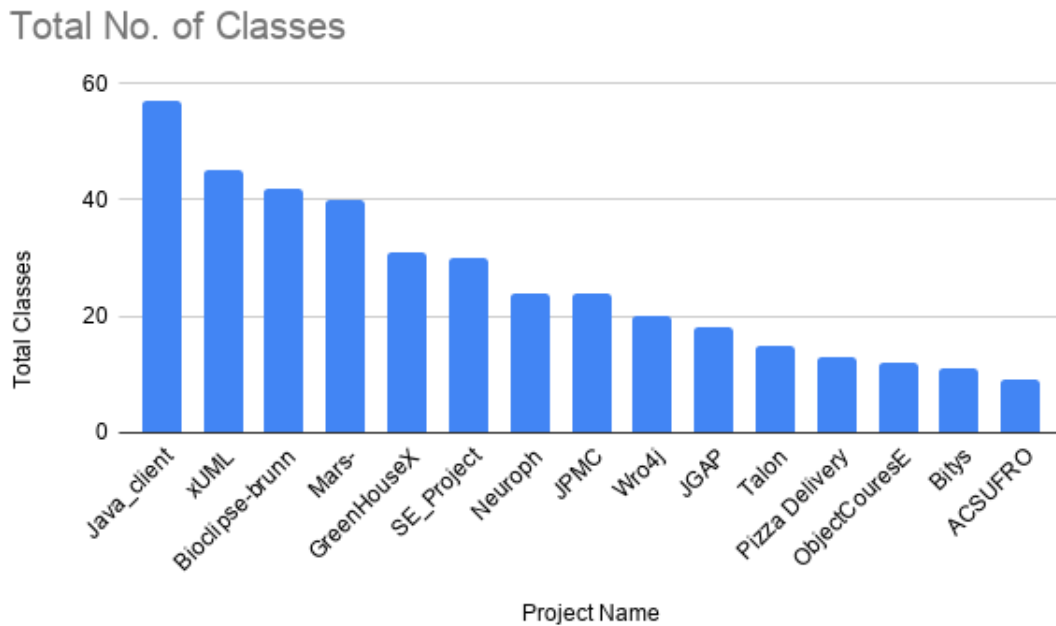


Figure 3.5: Total number of stereotypes

As a complementary view to the absolute table, we have created a graph that can visualize the data in the absolute table. As shown in figure 3.4, we can see that some projects such as Bitys have very limited role stereotypes as there are only Interfacer





**Figure 3.6:** *Total number of Classes for each project*

and Structurer identified in the class. And some projects such as Java\_Client and Mars-simulation, has one type of stereotype that was intensively found during the manual labeling.

#### 3.2.5.4 Distribution of Occurrence of absolute numbers of role stereotypes

Here we introduce the histogram to help to interpret the overall level of data. The histogram is used to show the distributional result of the provided data set. The x-axis shows the value of the result, in our case this should be the number of stereotypes we detected for each project. And as for y-axis, it is the number of projects that were identified with this number of stereotypes accordingly.

If we take a look into figure 3.7, we can see that the coordinator class in a number of 5-ish is identified all around the place with 15 projects. The Structurer (ST) class at value 1.00 to 2.00 is the second highest occurrences which can be found in around 13 projects. Followed up by Controller (CT) class at 4-5 and Interfacer (IF) class at around 5.56-7, they have been detected with 12 instances and 11 instances for the IF class. They both have another occurrence at higher value (CT=9, IF=10-11) but with lower than 5 projects to be detected. And there are around 6 projects that have no Information Holder class designed for their UML diagrams. The only high percentage stereotype can be found is Information Holder and the value is at 44.84%. There is only one project containing high ratio of IH like that. And last but not the least, the most often discovered range for percentages are between 0.00% to

around 13%.

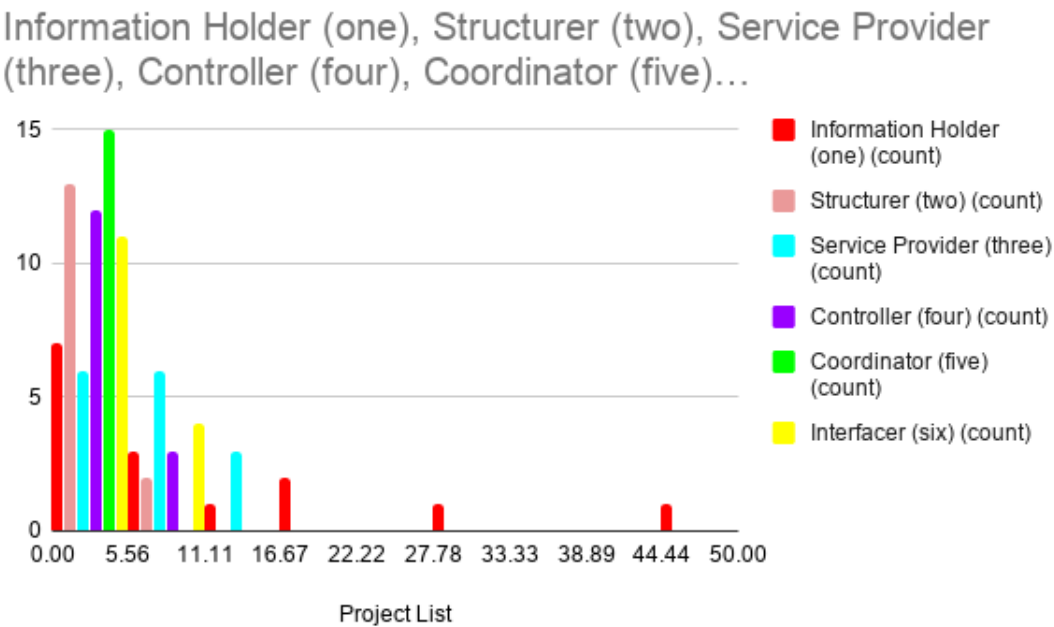


Figure 3.7: Histogram for absolute values

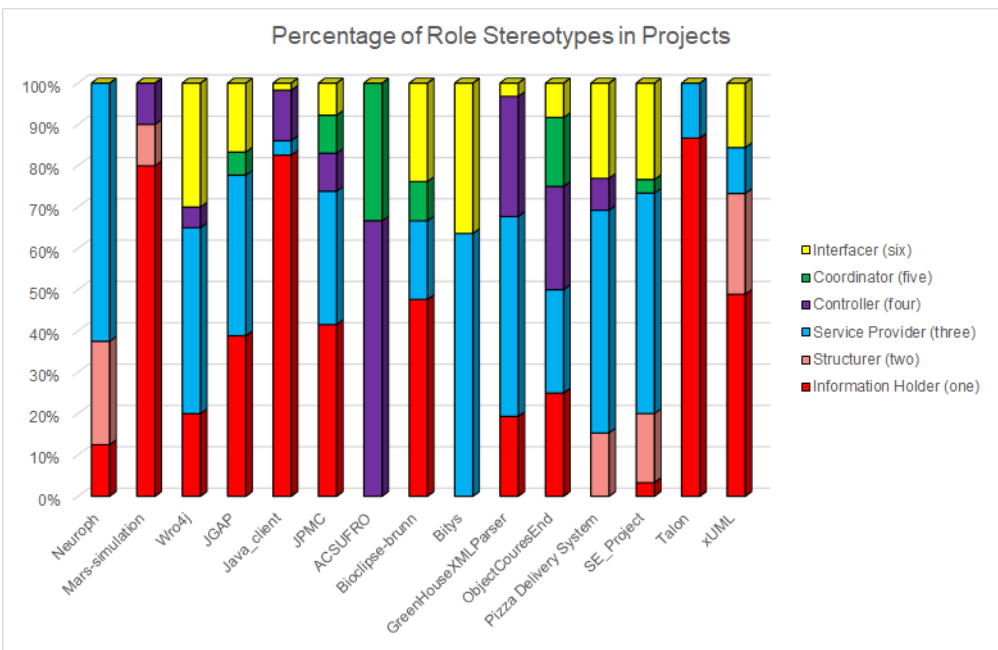


Figure 3.8: Percentage of role stereotypes for each project

### 3.2.5.5 Distribution of relative occurrence of role stereotypes

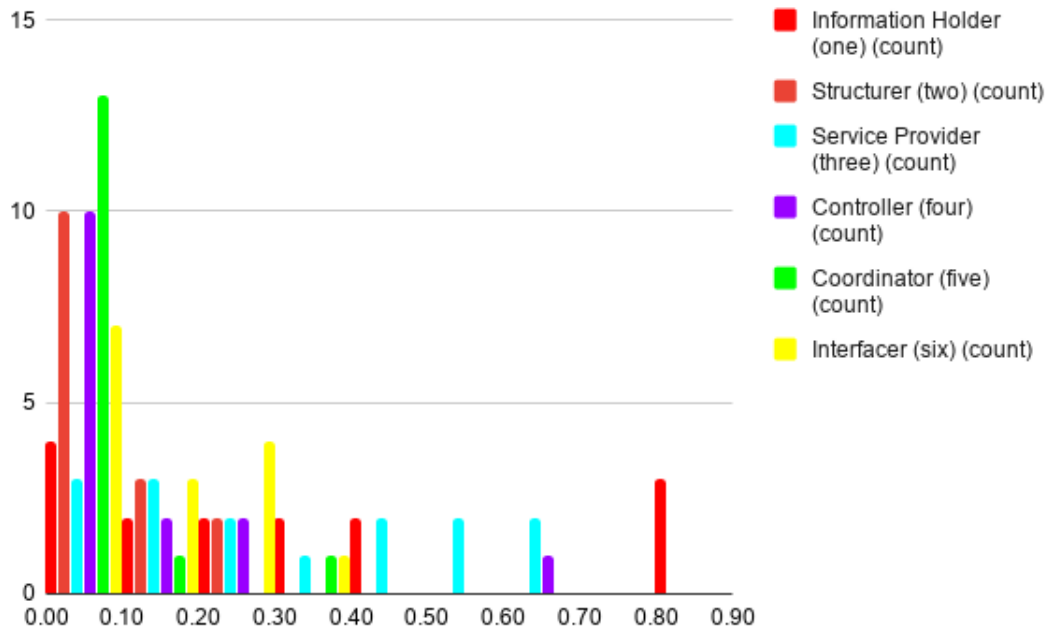
In order to get a more detailed view of the data set, we decided to further analyze the data. So we calculated the number of instances of table 3.2 and generated a table with percentage values. It serves a similar purpose of the absolute table but with a more high-end overview.

Percentage of role stereotypes for each project						
Project List	IH	ST	SP	CT	CD	IF
Neuroph	12.5%	25.0%	62.5%	0.0%	0.0%	0.0%
Mars-simulation	80.0%	10.0%	0.0%	10.0%	0.0%	0.00%
Wro4j	20.0%	0.0%	45.0%	5.0%	0.0%	30.0%
JGAP	38.9%	0.0%	38.9%	0.0%	5.6%	16.7%
Java_Client	82.4%	0.0%	3.5%	12.2%	0.0%	1.7%
JPMC	37.5%	0.0%	29.1%	8.3%	8.3%	16.6%
ACSUFRO	0.0%	0.0%	0.0%	66.7%	33.3%	0.0%
Bioclipse-brunn	47.6%	0.0%	19.0%	0.0%	9.5%	23.8%
Bitys	0.0%	0.0%	63.6%	0.0%	0.0%	36.4%
GreenHouseXMLParser	19.3%	0.0%	48.3%	29.0%	0.0%	3.2%
ObjectCourseEnd	25.0%	0.0%	25.0%	25.0%	16.7%	8.3%
Pizza_Deliver_System	0.0%	15.3%	53.8%	7.7%	0.0%	23.0%
SE_Project	3.3%	16.7%	53.3%	0.0%	3.3%	23.0%
Talon	86.7%	0.0%	13.3%	0.0%	0.0%	0.0%
xUML	48.9%	24.4%	11.1%	0.0%	0.0%	15.6%

**Table 3.3:** Percentage table for role stereotypes

In table 3.3, we can observe that the most common stereotype in Talon is Information Holder. It has a significant 86.70% among all other stereotypes in any other projects. The least found role stereotype is Service Provider which was identified with 0 occurrences within 8 projects. On the contrary, the most high frequency role stereotype is the Information Holder which has only 2 projects that were not detected with Information Holder class and it tends to exceed over 80% in at least 3 projects (Mars-sim, Java\_Client and Talon).

For the graph 3.8, we can see that many results conform to our previous observations such as the blue column, which is the Service Provider (SP) always takes up a certain portion within all of the projects that we have selected. As well as the Information Header (IH) has the highest frequency to be the most identified stereotypes. It has up to 98.7% in a project. Another notable findings from this graph is that not all stereotypes can be found within a project, we tend to find 3-4 types of role stereotypes within one project, and in some rare cases there are only 2-3 role stereotypes identified, e.g., Talon and Bitys. In the graph we can also see that the percentage of the role stereotypes are not evenly distributed, it diverges heavily to a direction that one or two stereotypes have the dominant partition in the project. Even in the projects that have almost all the role stereotypes, it still has this major



**Figure 3.9:** *histogram of the percentage of role stereotypes in projects*

role stereotype(s) that is close to the half ratio among the entire stereotypes' distribution. For example, in JPMC it has Information Holder identified with 37.50% which can be considered as a dominant attribute when we compare it with other roles: IF: 29.10%, CT: 8.30%, CD: 8.30% and ST: 16.66%.

In figure 3.9, we can see that the distributions of the each role stereotype and how many instances of them at which level of percentage. Take the Coordinator (CD) as an example, it shows a significant percentage distribution at 0.09 (9%) with around 13 projects. With a similar tendency, the Structurer (ST) can be found more often on the low percentages of the frequency, with 0%-10%, 10%-20% and 20%-30% on 10, 3, 2 projects respectively.

#### 3.2.6 Experiment with Machine Learning Algorithm

One of the most important machine learning (ML) prediction models is "classification technique" [24]. It can be described as the process of arranging the objects in groups or categories in a systematic way [25]. In our experiment we have used 4 machine learning algorithm which are, OneR, ZeroR, Random Forest (RF) and J48. These ML algorithms are really popular and broadly used in the software engineering area. For figuring out which ML algorithm works best these suite of algorithms are used.

In ML experiment it is quite important to build a baseline of performance using

a baseline classifier. It provides a point of reference which can be used to compare other ML algorithms performance. With this point of reference it is easier to see how much the performance has increased for other ML algorithms. In this case study, the OneR and ZeroR Algorithms are used for achieving that purpose.

**ZeroR** is the simplest classification methods, where it predicts only the majority category based on the training dataset. Although it doesn't have much predictability power, it can be used as a benchmark for other types of ML algorithms [26].

On the other hand, **OneR** is also a simple classification algorithm. Which produces a one-level decision tree. It generates one-rule for each of the attributes in the dataset and then it selects the rule which has minimum error as the "one-rule"[29]. In our experiments, we have used both the ZeroR and the OneR algorithms to determine a baseline performance for other algorithms.

In our experiments, we have used the gold standard 10-fold cross-validation [26]. In weka, the default value is 10-fold. We could increase or decrease the fold, as our dataset is medium large, the 10-fold was sufficient to measure the performance of our model. Another test option is *Percentage split*, which is used to train 66% of our dataset and the remaining 34% of the data will evaluate the performance of our model. We have measured the performance of our classification using TP (true positive) rate, FP (false positive) rate, precision, recall, F-measure and MCC (Matthews Correlation Coefficient).

We did two experiments for the ML algorithms we have mentioned earlier. In both experiments we have used all the role stereotypes as a separate classification category. As the classification category is more than two, it will be considered as Multi-class or Multi-role classification. For both experiment we have used 10-fold cross validation. In case of second experiment, we have used the dataset where we have considered all data which has "Confidence Level" attribute value from one to five. In case of 2nd experiment we have considered the dataset with highest value in the "Confidence Level" attribute, which is five. After running both experiments with the regular dataset, we have saved the results.

For improving the performance of our classifier and avoiding the overfitting problem we have used SMOTE (Synthetic Minority Over-sampling Technique) [27]. Overfitting occurs due to the imbalance in the dataset. It can happen when a trained classifier explains a given set of dataset but doesn't give the concrete rationale behind it [28]. One of the solution for overfitting problem is to train the classifier with more data, because of that we have decided to use SMOTE technique. So that we can do over-sampling of the minority classes and under-sampling of the majority classes. In both 1st and 2nd experiment we have used SMOTE after saving the results from the regular dataset. Later, we will do a comparison between the performance of each classifier which we will get from the regular dataset and using the SMOTE technique.

Next we have selected the ML algorithm, which performed best during the clas-

sification of our dataset. Then we have performed feature selection on our dataset, to check which attributes are more relevant to predict the role stereotypes.

#### **3.2.7 Analyze Classification Experiments**

In this section we will describe how we have analyzed the classification results of our experiments. We will run 2 experiments where each experiment will have 2 parts. In the first part of each experiment we will do a classification with the imbalanced dataset, in the second part we will run a classification after applying SMOTE resampling technique on the imbalanced dataset.

In each experiment, we will do an analysis of the confusion matrix, if 2 or more classifier have similar performance, then we will analyze and compare both for finding out the similarities and differences. Next, we will do a comparison among all the machine learning classifiers. Besides, we will run another experiment for finding out the ranking of the most relevant features for predicting the role-stereotypes. After analyzing the classification experiments, we will have a section where we will discuss our learning from those experiments and the case study in general.

# 4

## Results

In this chapter we present the result of our experiments. The result can be divided into two parts. In the first part we will discuss the result from our manual labeling and consolidation step (see figure: 3.1), which is the ground truth. The second part illustrates the outcome of our classification result by using machine learning algorithms. In details, several accuracy indicators such as Precision, Recall Rate are used in order to evaluate whether or not the determined result is reliable. The confusion matrix is analyzed in order to understand the faultiness and correctness in the distribution.

### 4.1 Evaluation of the Machine Learning Classifiers

In this section, we will describe the results of the experiments that we did for evaluating the classifiers that we built for classifying classes in UML class diagrams. In the section 3.2.5.2, we have mentioned that we have 2 datasets and we have used the label "Confidence Level" to differentiate between them. As we have low number of classes in each project, we have considered all the classes from the selected 15 projects together in each dataset 3.1. In the following sections, we will describe the classification results that we got after applying ML algorithm on those two datasets.

#### 4.1.1 Multi-class classification for All *Confidence Level*

In this experiment, we have considered classes which are labelled with one to five confidence level, that means all level in the Confidence Level scale are considered here. Our dataset consists of 391 classes from 15 projects (see table 3.1). In the following table 4.1, we have displayed the result for different ML algorithms. The first four rows are showing the results for the baseline classifiers, and the last four rows are showing the result for in-depth classifiers. First we have run classifier on the regular imbalanced dataset, and next we have applied SMOTE resampling technique on the dataset to see if the accuracy has improved. In the table, we have written "SMOTE" beside the classifier name, when it is not the regular imbalanced dataset. We can see from the table 4.1 that the classifiers performed best when we have applied SMOTE for balancing the imbalanced dataset.

#### 4. Results

Multi-roles Classification Result (All Confidence Level)						
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
OneR	0.427	0.427	0.000	0.427	0.000	0.000
ZeroR	0.427	0.427	0.000	0.427	0.000	0.000
OneR (SMOTE)	0.821	0.040	0.909	0.821	0.837	0.818
ZeroR (SMOTE)	0.227	0.227	0.000	0.227	0.000	0.000
J48	0.637	0.165	0.627	0.637	0.628	0.483
Random Forest	0.670	0.224	0.743	0.670	0.635	0.526
J48 (SMOTE)	0.736	0.077	0.000	0.736	0.000	0.000
Random Forest (SMOTE)	0.904	0.024	0.905	0.904	0.904	0.882

**Table 4.1:** Classification Results for All Confidence Level

For the imbalanced dataset, the TP rate, FP rate, recall are similar for both OneR and ZeroR classifier, which are the benchmark for the other 2 classifiers. The Precision, F-measure and MCC values are unavailable for OneR and ZeroR. On the other hand, all the values in the accuracy table 4.1 are present for J48 and Random Forest (RF) classifier and RF performed better than the J48 classifier. So, we can conclude that RF is the best performing classifier for the imbalanced dataset.

In case of dataset after applying SMOTE, we can see from the accuracy table 4.1 that, the values have changed a lot. For ZeroR classifier, the TP rate, FP rate and Recall values are reduced from the benchmark. On the other hand, the values for OneR, J48 and RF has increased a lot. In this case, Precision, F-measure and MCC are unavailable for ZeroR and J48. According to the result, the Random Forest classifier has achieved the highest score for the dataset with SMOTE. For getting a better picture of the classification result, an illustration has been shown in figure 4.1.

	IH	ST	SP	CT	IF	CO
IH	158	0	9	0	0	0
ST	23	4	1	0	0	0
SP	39	0	63	0	1	0
CT	10	0	1	22	0	0
IF	26	1	6	0	14	0
CO	10	0	2	0	0	1

**Table 4.2:** Confusion Matrix for Random Forest classifier for the imbalanced dataset (All confidence level)

	IH	ST	SP	CT	IF	CO
IH	134	10	17	1	5	0
ST	14	10	3	1	0	0
SP	32	1	54	3	6	7
CT	1	0	5	26	1	0
IF	14	2	8	2	21	0
CO	2	0	6	0	1	4

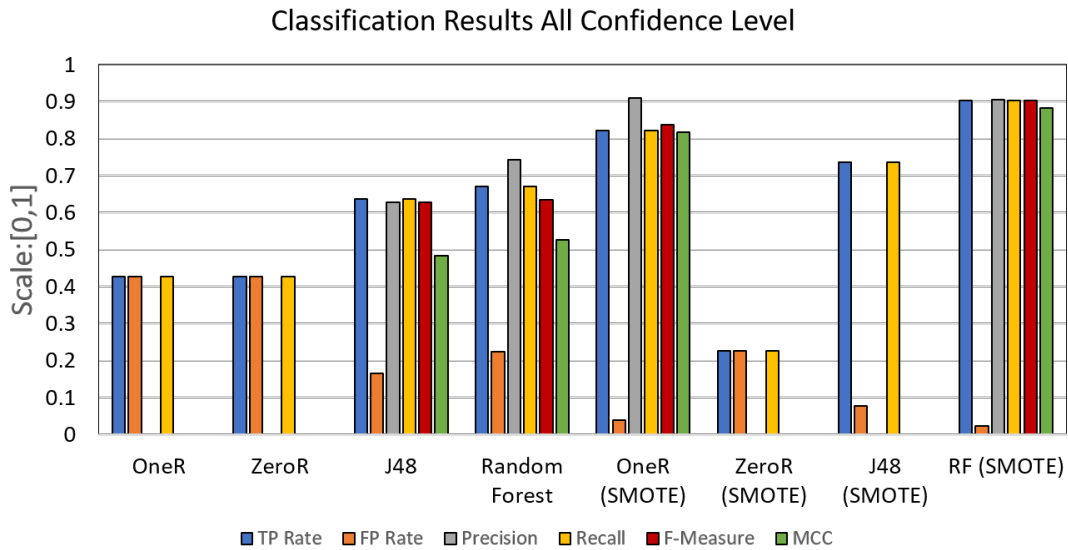
**Table 4.3:** Confusion Matrix for J48 classifier for the imbalanced dataset (All confidence level)



We have shown the confusion matrix for the Random Forest (RF) and J48 classifier on the imbalanced dataset in the figure 4.3 figure 4.2 sequentially. The best performing fold was selected from the 10-fold cross-validation. The first column and the first row are showing the actual and predicted role-stereotypes consecutively. In this case, the classification accuracy of RF and J48 are quite similar. In the next paragraph we will do a comparison to see which classifier performed best.

In case of RF, if we look at the diagonal of the confusion matrix we can see that 262 classes are classified correctly out of the 391 classes. Due to the imbalance in the dataset, the classifier misclassified 129 classes. The RF classifier has classified 158 Information Holders correctly out of 167. It has classified 63 Service Providers correctly, out of 103. The current accuracy for RF classifier is 67%, for this imbalanced dataset. After applying the SMOTE in the dataset, the accuracy for RF classifier has increased to 90.43%.

In case of J48 classifier, when we look at the diagonal of the matrix we can see that, the classifier has correctly classified 249 instances and it classified 142 instances incorrectly. It has classified 134 information holders properly out of 167, which is less, comparing to the RF classifier. In this case the classifier classified 54 service providers out of 103, which is less than RF. Other than that, J48 classified structurer, controller, coordinator and interfacier more accurately than RF. The current accuracy for J48 classifier is 63.68%, for this imbalanced dataset. After applying the SMOTE in the dataset, the accuracy for J48 classifier has increased to 73.59%. That means, the imbalance behavior in the dataset can affect the performance of the classifiers. By seeing the comparison, we can say that RF classifier has performed best when classifying the imbalanced dataset for All Confidence Level.



**Figure 4.1:** *Illustration of Classification Results for All Confidence Level*

### 4.1.2 Multi-class classification for High *Confidence* Cases

In this experiment, we have considered the dataset where classes are labelled with only the high confidence level which is five. This time the dataset consists of 328 classes from the 15 projects (see table 3.1). In the following table 4.4, we have displayed the result for different ML algorithms. The first four rows are showing the results for the baseline classifiers, and the last four rows are showing the results for in-depth classifiers. First we have run classifier on the regular imbalanced dataset, and next we have applied SMOTE technique on the dataset to see if the accuracy has improved. In the table, we did not write anything beside the classifier name, when it is a regular imbalanced dataset, otherwise we have written "SMOTE" beside the classifier name.

In the imbalanced dataset, the TP rate, FP rate, recall are similar for both OneR and ZeroR classifiers, which are the benchmark for the other 2 classifiers. The Precision, F-measure and MCC values are unavailable for OneR, ZeroR and Random Forest (RF). In case of J48 classifier, all the values in the accuracy table 4.4 are available. The TP rate, FP rate and Recall values of Random Forest (RF) classifier is a bit higher than the values of J48 classifier. Still we can say that J48 classifier classifies the imbalanced dataset more precisely, as it has all the values at hand. So, we can draw the conclusion by saying that, J48 classifier is the best performing classifier for the imbalanced dataset.

Multi-roles Classification Result (High Confidence Level)						
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
OneR	0.442	0.442	0.000	0.442	0.000	0.000
ZeroR	0.442	0.442	0.000	0.442	0.000	0.000
OneR (SMOTE)	0.776	0.059	0.000	0.776	0.000	0.000
ZeroR (SMOTE)	0.223	0.223	0.000	0.223	0.000	0.000
J48	0.683	0.154	0.669	0.683	0.669	0.543
Random Forest	0.695	0.222	0.000	0.695	0.000	0.000
J48 (SMOTE)	0.847	0.039	0.900	0.847	0.850	0.834
Random Forest (SMOTE)	0.881	0.030	0.891	0.881	0.878	0.858

**Table 4.4:** Classification Results for High Confidence Level

In case of dataset with SMOTE, we can see from the table 4.4 that, the values for OneR, J48 and Random Forest (RF) classifiers have increased. On the other hand, the values of ZeroR classifier in the accuracy table 4.4 has been reduced. Precision, F-measure and MCC values are missing for both OneR and ZeroR classifiers. In this case, both J48 and RF have all the values in the accuracy table 4.4. For RF

the TP rate, recall, F-measure and MCC have higher values than the corresponding J48 classifier values. So, we conclude by saying that, RF classifier has achieved the best score for classifying the dataset with SMOTE. For getting an overview of the classification result, an illustration has been portrayed in the figure 4.2.

	IH	ST	SP	CT	IF	CO
IH	139	0	5	0	1	0
ST	20	4	1	0	0	0
SP	29	0	54	0	0	0
CT	8	0	1	21	0	0
IF	25	0	4	0	10	0
CO	4	0	2	0	0	0

**Table 4.5:** Confusion Matrix for RF classifier for the imbalanced dataset (High confidence level)

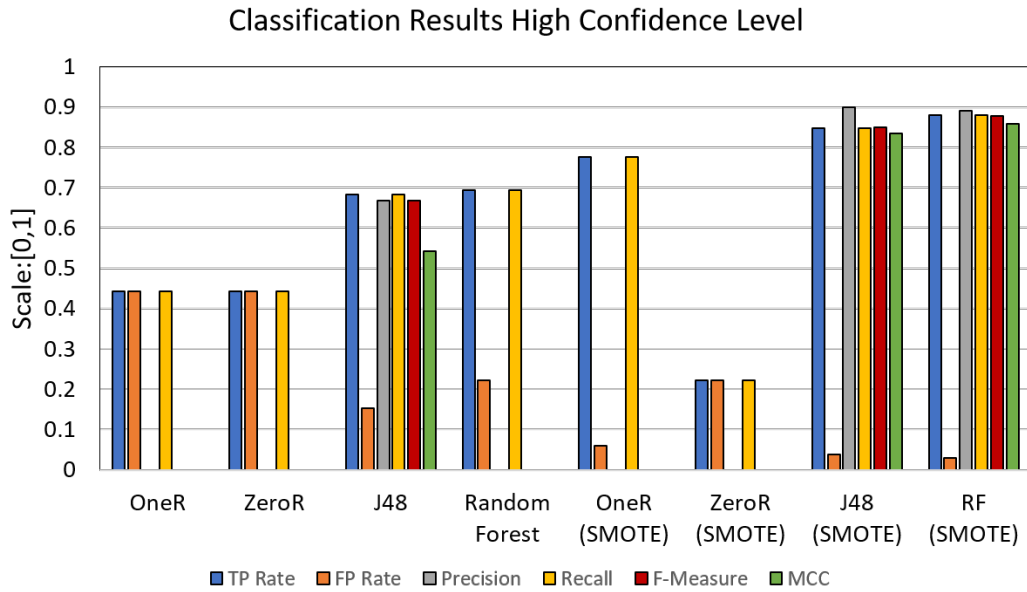
	IH	ST	SP	CT	IF	CO
IH	123	8	9	1	3	1
ST	14	8	2	1	0	0
SP	19	2	53	0	7	2
CT	1	1	1	26	1	0
IF	14	1	12	0	11	1
CO	1	0	2	0	0	3

**Table 4.6:** Confusion Matrix for J48 classifier for the imbalanced dataset (High confidence level)

We have illustrated the confusion matrix for the Random Forest (RF) and the J48 classifiers on the imbalanced dataset in the figure 4.6 and figure 4.5 consecutively. We have selected the best performing fold out of the 10-fold cross-validation. The first column and the first row are showing the actual and predicted role-stereotypes in a successive manner. As the performance of RF and J48 classifiers are almost similar, we will see the similarities and the differences in the following 2 paragraphs.

Looking at the values for RF classifier in the confusion matrix, we can see that, the diagonal of the matrix represents the number of classified classes, which is 228. Outside the diagonal there are 100 classes, which are incorrectly classified. Here, the RF classifier has classified 139 Information Holders out of 145 of them, which is quite accurate. On the other hand, RF classifier has failed to classify any Co-ordinators. In this dataset, the number of coordinators are really low, which is 6, it could be a reason for the failure. Low number means, the classifier can look at few examples to train itself. The accuracy of the RF classifier during classifying the imbalanced dataset is 69.51%. When we have applied SMOTE resampling technique on the dataset, the accuracy of RF classifier has been increased to 88.14%.

Next we will look at the values of J48 classifier in the confusion matrix. From the diagonal of the confusion matrix, we can see that 224 instances are classified correctly, whereas it failed to correctly classify 104 instances or classes. In this case, the J48 classifier has correctly classified 123 Information Holder out of 145 of them, that means J48 is a bit inaccurate than RF during classifying the Information Holder. On the contrary, J48 classifier has successfully classified 3 coordinators out of 6. The accuracy of the J48 classifier during classifying the imbalanced dataset is 68.29%. When we have applied SMOTE resampling technique on the dataset, the accuracy of J48 classifier has been increased to 84.71%. By seeing the similarities and differences we can say that J48 classifier has performed slightly better than the RF classifier during classifying the imbalanced dataset.



**Figure 4.2:** *Illustration of Classification Results for High Confidence Level*

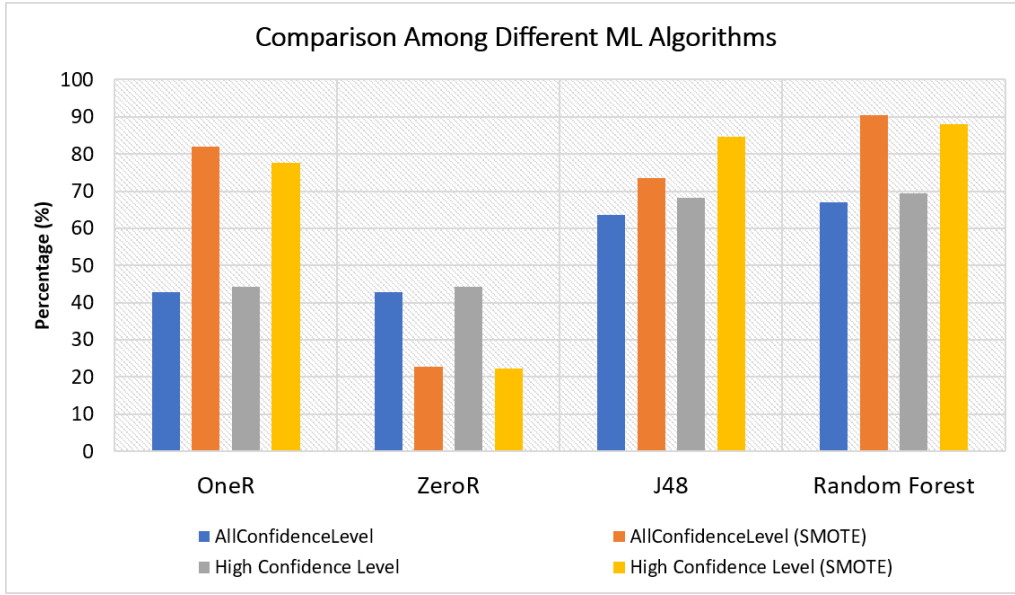
## 4.2 Comparing Performance of different ML algorithms

In this section, we have discussed the comparisons among different classifiers performance. By comparing the classification results between the table 4.1 and the table 4.4, we can say that the dataset with high confidence level is classified more accurately than the dataset with all confidence level values. After applying the SMOTE balancing technique on the dataset, the classification accuracy of the classifiers has increased even more. In the table 4.7, we have displayed the performance results for different classifiers that we have considered. For better understanding an overview has been illustrated in the figure 4.3.

Comparisons Among Different Classifiers Accuracy (in percentage)				
Algorithm Names	All Confidence Level	All Confidence Level (SMOTE)	High Confidence Level	High Confidence Level (SMOTE)
OneR	42.71	82.07	44.21	77.57
ZeroR	42.71	22.66	44.21	22.29
J48	63.68	73.59	68.29	84.71
Random Forest	67.00	90.43	69.51	88.14

**Table 4.7:** Comparisons among different classifiers accuracy

From the table 4.7, we can see that the two baseline classifiers OneR and ZeroR acted



**Figure 4.3:** *Comparison Among Different Machine Learning Algorithms Performance*

differently, when we have applied SMOTE. After applying SMOTE the performance of OneR has increased and the performance of ZeroR has decreased. By comparing the two table, table 4.1 and table 4.4, we can draw a conclusion by saying that, the RF classifier performed best when classifying the dataset with all Confidence Level, on the other hand J48 classifier performed best when classifying the dataset with High Confidence Level. Performance distribution or classification accuracy of the 4 classifiers on different datasets are added in the appendix C.

#### 4.2.1 Deeper analysis of best performing classifiers

In this section, we will describe the classification accuracy for each role-stereotype by J48 and Random Forest classifiers. In table 4.8, 4.9, 4.10 and 4.11, we have shown the classification accuracy for J48 while classifying the six role-stereotypes namely, Information Holder (IH), Structurer (ST), Service Provider (SP), Controller (CT), Interfacer (IF) and Coordinator (CO). In this section we will describe the four accuracy table that we have produced during the classification experiment with J48 classifier.

The first two table illustrates the classification accuracy on imbalance dataset with All Confidence Level <sup>2</sup> and the High Confidence Level consecutively. The rest two table illustrates the results for imbalanced dataset with All and High confidence level sequentially, in this case we have ran those experiment after applying SMOTE resampling technique.

<sup>2</sup>**Confidence Level** is a label which has been introduced by us, for measuring the confidence during the prediction of role stereotypes. The scale for the label is one to five, where five is the highest confidence level

#### 4. Results

Similarly we have described the table 4.12, 4.13, 4.14 and 4.15 for the RF classifier. The first two table illustrates the classification accuracy on imbalance dataset with All Confidence Level and the High Confidence Level successively. The rest two table illustrates the results for imbalanced dataset with All and High confidence level consecutively, in this case we have ran those experiment after applying SMOTE resampling technique.

We can see from table 4.8 that J48 is really good at detecting the IH and CT classes when classifying the dataset with All Confidence Level. Besides, we can see from the table that J48 is medium good at detecting the SP, IF classes. On the contrary, it was not that good when it comes to classifying the CO and ST classes.

J48 classifier - Imbalance dataset - All Confidence Level						
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
IH	0.802	0.281	0.680	0.802	0.736	0.516
ST	0.357	0.036	0.435	0.357	0.392	0.352
SP	0.524	0.135	0.581	0.524	0.551	0.402
CT	0.788	0.020	0.788	0.788	0.788	0.768
IF	0.447	0.038	0.618	0.447	0.519	0.472
CO	0.308	0.019	0.364	0.308	0.333	0.314

**Table 4.8:** Accuracy table for J48 Classifier - All Confidence Level

From table 4.9, we can see that, J48 is better when it comes to detecting IH and CT classes when classifying the dataset with High Confidence Level. The SP and CO classes falls into the second category, as J48 was medium good when detecting them. We can add the ST and IF classes in the last category as the number of TP rate is really low for these two cases.

J48 classifier - Imbalance dataset - High Confidence Level						
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
IH	0.848	0.268	0.715	0.848	0.776	0.577
ST	0.320	0.040	0.400	0.320	0.356	0.311
SP	0.639	0.106	0.671	0.639	0.654	0.541
CT	0.867	0.007	0.929	0.867	0.897	0.887
IF	0.282	0.038	0.500	0.282	0.361	0.316
CO	0.500	0.012	0.429	0.500	0.462	0.452

**Table 4.9:** Accuracy table for J48 Classifier - High Confidence Level

Table 4.10 shows that, J48 classifier failed to classify the IH role stereotype when classifying the dataset with All Confidence Level and we have run the experiment after applying SMOTE resampling technique on the dataset. According to the table J48 classified SP, CT and CO better than other stereotypes. The classifier was medium good during the detection of ST and IF.

J48 classifier - dataset with SMOTE - All Confidence Level						
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
IH	0.000	0.000	0.000	0.000	0.000	0.000
ST	0.786	0.000	1.000	0.786	0.880	0.873
SP	1.000	0.341	0.462	1.000	0.632	0.552
CT	0.939	0.000	1.000	0.939	0.969	0.964
IF	0.830	0.000	1.000	0.830	0.907	0.891
CO	0.913	0.000	1.000	0.913	0.955	0.950

**Table 4.10:** Accuracy table for J48 Classifier - All Confidence Level (SMOTE)

J48 classifier - dataset with SMOTE - High Confidence Level						
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
IH	0.986	0.182	0.586	0.986	0.735	0.684
ST	0.810	0.000	1.000	0.810	0.895	0.886
SP	0.446	0.006	0.902	0.446	0.597	0.605
CT	0.942	0.000	1.000	0.942	0.970	0.965
IF	0.833	0.000	1.000	0.833	0.909	0.892
CO	0.927	0.003	0.978	0.927	0.952	0.945

**Table 4.11:** Accuracy table for J48 Classifier - High Confidence Level (SMOTE)

From the table 4.11 we can see that, J48 classifier performed best when detecting IH, CT and CO during the classification of the dataset with High Confidence Level data. We have ran this experiment after applying SMOTE technique on the dataset. The classifier was medium good when detecting the ST and IF. It performed poorly when detecting the SP role stereotype.

Random Forest classifier - regular imbalanced dataset - All Confidence Level						
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
IH	0.946	0.482	0.594	0.946	0.730	0.492
ST	0.143	0.003	0.800	0.143	0.242	0.322
SP	0.612	0.066	0.768	0.612	0.681	0.590
CT	0.667	0.000	1.000	0.667	0.800	0.804
IF	0.298	0.003	0.933	0.298	0.452	0.499
CO	0.077	0.000	1.000	0.077	0.143	0.273

**Table 4.12:** Accuracy table for Random Forest Classifier - All Confidence Level

Table 4.12 represents that RF classifier performed best when detecting IH, CT and SP classes when classifying the imbalanced dataset with all confidence level classes. Besides, from this table we can see that RF is medium good when it comes to identifying the ST and IF classes. The RF classifier performed poorly when identifying the CO classes.

Random Forest classifier - regular imbalanced dataset - High Confidence Level						
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
IH	0.959	0.470	0.618	0.959	0.751	0.523
ST	0.160	0.000	1.000	0.160	0.276	0.387
SP	0.651	0.053	0.806	0.651	0.720	0.644
CT	0.700	0.000	1.000	0.700	0.824	0.824
IF	0.256	0.003	0.909	0.256	0.400	0.455
CO	0.000	0.000	0.000	0.000	0.000	0.000

**Table 4.13:** Accuracy table for Random Forest Classifier - High Confidence Level

According to the table 4.13, Random Forest (RF) discovered the IH, CT, and SP classes quite accurately when classifying the imbalanced dataset with high confidence cases. We can also notice from this table that, RF performed fairly good when discovering the IF and ST classes. On the other hand, the RF classifier didn't perform well enough when spotting the CO classes.

Random Forest classifier - dataset with SMOTE - All Confidence Level						
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
IH	0.778	0.038	0.823	0.778	0.800	0.757
ST	0.884	0.008	0.943	0.884	0.912	0.901
SP	0.917	0.038	0.875	0.917	0.896	0.865
CT	0.955	0.000	1.000	0.955	0.977	0.973
IF	0.952	0.032	0.886	0.952	0.918	0.897
CO	0.952	0.004	0.971	0.952	0.961	0.956

**Table 4.14:** Accuracy table for Random Forest (RF) Classifier - All Confidence Level (SMOTE)

From table 4.14, we can see that Random Forest (RF) classifier performed better when identifying the SP, CT, IF and CO classes. In this case, we have run the experiment on the dataset after applying SMOTE resampling technique and we have considered All Confidence Level. Besides, from this table we can see that RF is medium good when it comes to identifying the IH and ST classes.

Random Forest classifier - dataset with SMOTE - High Confidence Level						
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC
IH	0.883	0.086	0.727	0.883	0.798	0.744
ST	0.870	0.007	0.956	0.870	0.911	0.898
SP	0.530	0.008	0.898	0.530	0.667	0.661
CT	0.967	0.003	0.983	0.967	0.975	0.970
IF	0.955	0.042	0.866	0.955	0.909	0.882
CO	0.969	0.002	0.989	0.969	0.979	0.976

**Table 4.15:** Accuracy table for Random Forest Classifier - High Confidence Level (SMOTE)



According to the table 4.15, Random Forest (RF) discovered the CT, IF and CO classes quite accurately when classifying the dataset after applying SMOTE. In this case, we have considered the high values of Confidence Level metrics. We can also notice from this table that, RF performed fairly good when discovering the IH and ST classes. On the contrary, the RF classifier didn't perform well enough when detecting the SP classes.

### 4.3 Ranking of Relevant Features for predicting role-stereotypes

In this section, we have demonstrated the relevant features for predicting the role-stereotypes in the UML class diagrams. In the previous section 4.2, we have seen that the classifiers performs best on the dataset with only "High Confidence Level" data. That is why we have run feature selection on the dataset with "High Confidence Level" data. Before running the feature selection, we have selected the "select attribute" tab from the explorer tab of weka. Then, we have selected the *CorrelationAttributeEval* option from the *Attribute Evaluator* pane. We have used *Ranker* search, as it was the recommended search with this feature selection method. We ran this feature selection on two dataset. One dataset is the regular one with only "High Confidence Level" data, and the other one is "High Confidence Level" data with SMOTE. The comparison between these two results are presented in the table 4.16.

From the table 4.16, we can see that the attributes are listed from higher to lower ranked value. The attribute with larger number in the ranked column means, it is more relevant for predicting the role-stereotypes. Besides table 4.16 shows that, when we have added SMOTE in the dataset, the ranking has changed for many attributes. There are a lot of similarities and differences in both outcome of the feature selection method.

In both results, the attribute "EndWithManager,Controller" has the largest value, this metrics was added by us. We have created this metrics for detecting the Controller (CT) classes. The higher rank means, it is a deciding factor when labeling the Controller (CT) classes (check 3.2.3 & 3.2.4). Two other interesting attributes are, "IsEntity" and "Getters", which are the deciding factor when labeling the Information Holder (IH) classes. The attribute "EndWithFactory, Impl,Implementation" is a deciding factor while labeling the Service Provider (SP), although its rank is not higher in the results, it played a important role while labeling the role stereotypes. Another attribute "HasType,Annotation, List" is also an important factor when labeling the Information Holder, in both results it has a lower ranked value. Lower ranked value means, the dataset doesn't have that many classes which is related to "HasType,Annotation, List" attribute. For other attributes the ranked has shown drastic changes, when we have run feature selection method on the dataset with SMOTE.

Comparison Between Feature Selection			
Relevant Attributes in Regular Dataset		Relevant Attributes in Dataset with SMOTE	
Attributes	Ranked	Attributes	Ranked
EndWithManager, Controller	0.246	EndWithManager, Controller	0.309
OpsInh	0.228	IsEntity	0.182
DIT	0.222	CLD	0.171
Getters	0.204	Getters	0.139
NumAnc	0.201	NumAssEl__ssc	0.135
CLD	0.170	NOC	0.134
NumPubOps	0.157	OpsInh	0.131
NumAttr	0.149	NumDesc	0.130
NumOps	0.147	DIT	0.125
AttrInh	0.143	NumOps	0.123
NOC	0.141	IFImpl	0.122
Setters	0.136	Nesting	0.119
NumDesc	0.128	Setters	0.117
EC_Attr	0.121	NumPubOps	0.114
IFImpl	0.121	NumAssEl__nsb	0.111
IC_Par	0.117	NumAssEl__sb	0.109
NumAssEl__sb	0.113	Name	0.107
NumAssEl__ssc	0.099	IC_Par	0.101
Dep_Out	0.099	NumAnc	0.099
Dep_In	0.098	EC_Attr	0.099
Nesting	0.095	IC_Attr	0.098
IC_Attr	0.091	EC_Par	0.094
HasType, Annotation, List	0.090	Dep_Out	0.089
EndWithFactory, Impl, Implementation	0.087	AttrInh	0.088
EC_Par	0.085	Dep_In	0.086
NumAssEl__nsb	0.067	NumAttr	0.085
IsEntity	0.066	HasType, Annotation, List	0.060
Name	0.046	EndWithFactory, Impl, Implementation	0.043

**Table 4.16:** Comparison between Dataset with Regular data and Dataset with SMOTE

## 4.4 Discussion

In this section, we will discuss our findings and the learning outcome of this case study. The classifier that we have built during the course of this case study was designed by keeping the software design and UML class diagrams in mind (check 2.1 & 2.2). Our classifier has a simple design and easy to implement. Throughout this case study several decisions were taken, based on those decisions we have updated the criteria of selection and other parts of the case study. A lot of time and effort was given during the manual labeling of the classes in UML class diagrams. We have repeated each steps in the methodology several times to establish and refine our ground truth.

As we have mentioned earlier in section 3.2.4, that we have used the six role-stereotypes from Wirf-brocks research [1] to label the classes in UML class diagrams. Role stereotypes were useful during the investigation of the design pattern of UML class diagrams. Chaudron et al. demonstrated how they have used role-stereotypes for labeling the classes in Source Code [3]. In another case study, Dragan et al. has mentioned how useful role-stereotypes are in case of source code, besides they have mentioned the term role-stereotypes in UML [7]. In our case study, we have used role-stereotypes for labeling the UML class diagrams. We have found that, there are several important factors which plays key role at the time of deciding the role-stereotype of a certain class. Class names, method names and type of dependencies are some of the important factor that we have checked while labeling the classes from UML class diagrams.

In section 3.2.3, we have demonstrated how we have extracted features or metrics to use during the Machine Learning (ML) experiment. In the beginning we have used only the metrics that we got from the SD Metrics tool. Later, we have discussed these results with our thesis supervisor and other experts, they have suggested us to introduce new metrics based on our experience from those experiment. We have created some features by looking at the class names which are repetitive, method names and other characteristics which indicates certain role-stereotype. After we have added our own created features, the ML classification performance has been improved immensely. This way we have learned how adding nominal types of features which are relevant to the UML class diagrams, can affect the performance of a ML algorithm.

### 4.4.1 Insights from the Ground truth - Manual labeling

In our manual labeling stage, we defined our ground truth which aimed to shape the scope of our research study. In the meanwhile, we have received our most valuable result from this stage. As shown in section 3.2.5.3 in table 3.2 and 3.3, we can see that there are many empty values for the Structurer (ST) in our projects (>9). However, the total number of ST classes are not the least comparing to the Coordinator Classes. This could simply mean that the ST type of classes are aimed to help to define a project's architecture, which should "delegate" some parts of the

work of the system to the outside world[1][9]. However, all of these projects that are missing with the ST seem to be either with less than 10 classes, e.g., ACSOFRO (9) and ObjectCourseEnd (12) or it is more of a utility-based project such as Wro4j and JGAP.

As noted in table 3.2, the highest value was detected in Java\_Client for role Information Holder. In fact, if we look at the total number of classes in this project we can find that this project takes up to 82.40% of the entire role stereotypes in this project. There are a handful of other roles but they are much less significant to be analyzed in this case. So we did a dig into the project and examined the provided UML class diagram. We found that most of the classes that are labelled IH or Interfacer (IF) has many attributes which contain data and a few methods that reads/retrieves/transfers these data to another place. Thus we assume that when a project has a high number of IH classes, it tends to have a strong data flow/exchange within the project.

Another thing is that, in figure 3.5 it displays the overall distribution of all 6 role stereotypes in all of our selected projects. From this distribution, we can observe that IH has the highest percentage of occupancy, with 42.7% identified rate. The least detected role is Coordinator (CD) with only 3.30%. The Structurer (ST) and Controller (CT) follows up the CD closely with 7.20% and 8.40% which are both lower than 10%. As for the Interfacer (IF) and Service Provider (SP) they have decent identified rate with 12.% on IF and 26.30% which is the second highestly identified role stereotypes after IH. What we can see with this data is that a decent software project is formed with classes that can define its data structurer. Then it should have a decent amount of SP as every software should some meanings in "Do something". Therefore the SP is one necessary role stereotype that can do these.

### 4.4.2 Comparison between role-stereotypes in UML Class diagrams vs in Source Code

In this section we will describe the similarities and differences between our classifier and the source code classifier from Chaudron's et al. paper [3]. Our discussion will be based on two perspective, they are manual labeling perspective and machine learning perspective.

#### 4.4.2.1 Manual Labeling perspective

In this section, we will compare our classifier URI (UML role identifier) with the CRI (Source Class Role Identifier) [3], which is shown in table 4.17. Chaudron et al. has presented CRI in his research, which classifies class role-stereotypes in source code. They have established their ground truth based on 779 labelled classes selected from source code (based on manual labeling) [3], and in our case study we have established our ground truth based on 391 classes discovered from UML class diagrams (Based on our manual labeling). Both our studies and their studies has considered the six role-sterotypes from Wirfs-Brock research [1], for labeling the

classes.

From this point on, we will write SrcRI to refer to Chaudron's Source code Classifier (called CRI in their paper) order to make the abbreviations easier to recognize as referring to source code (and URI to refer to our UML-level class-classifier). In this comparison, we have calculated the frequencies of role-stereotypes in percentage, as the number of classes are different in both case study. As we have used percentage instead of the absolute number of the classes, the reader can comprehend the differences between URI and SrcRI more easily. In table 4.17, we have four column, the first column represents the six role-stereotypes, second and third column represents the percentages of role-stereotypes occurrences in our 2 datasets (All and High "Confidence Level"<sup>4</sup>) and the fourth column represents the percentages of role-stereotypes occurrences in source code. The fourth column values are adapted from Chaudron et al. case study, which can be found in figure 4 from their case study [3].

Comparison between URI and SrcRI (frequencies of role-stereotypes as %)			
<i>Role Stereotypes</i>	<i>All Confidence Level (URI)</i>	<i>High Confidence Level (URI)</i>	<i>SrcRI</i>
Information Holder	42.71	44.21	29.7
Structurer	7.16	7.62	6.3
Service Provider	26.34	25.30	41.5
Controller	8.44	9.15	2.5
Coordinator	3.32	1.83	10.1
Interfacer	12.02	11.89	9.9
Total no. of classes (absolute value)	391	328	779

**Table 4.17:** Comparison between URI and SrcRI classifiers

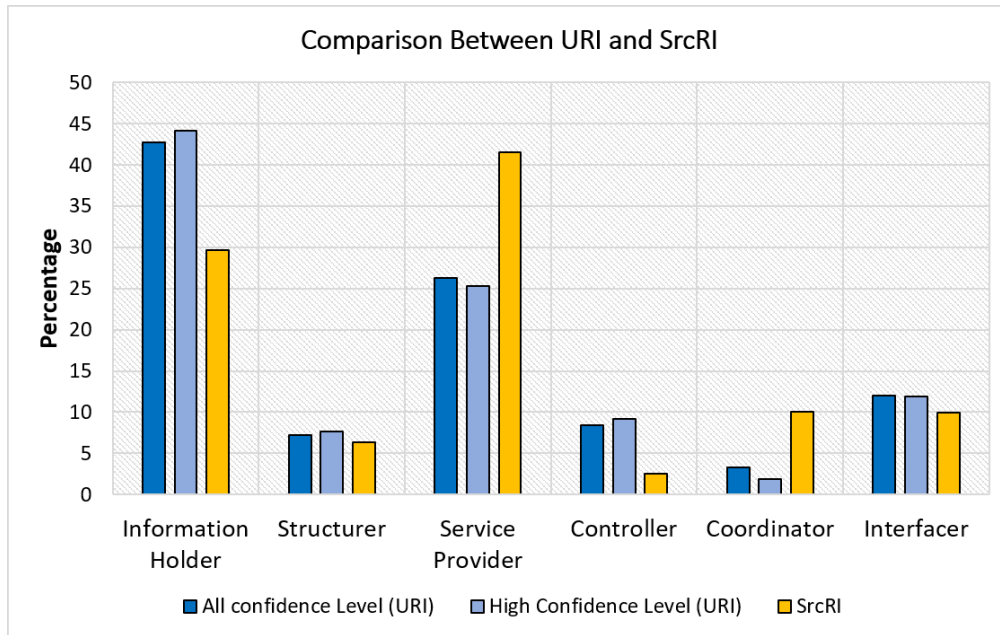
We can see from table 4.17, the 2nd and 3rd column values are quite similar as the data-sets are classified by the same URI classifier. By seeing the comparison, we can say that "Information holder" role-stereotype is more visible ([42.71%,44.21%]) when classifying with URI, on the other hand SrcRI detected "Service provider" role-stereotype more often (41.5%) when classifying the source code. Another interesting fact about the two classifiers is that, "Information Holder" and "Service Provider" occurs predominantly than other role-stereotypes when classifying with URI and CRI. In case of URI, the 3rd role stereotype which is predominant is "Interfacer" ([11.89%,12.02%]), on the contrary for SrcRI the 3rd predominant role-stereotype is Coordinator (10.1%). In both cases, the percentage of occurrences for "Structurer" role-stereotype is quite similar.

Another noticeable fact in table 4.17, the frequency of controllers and coordinators. In case of URI, controller ([8.44,9.15]) has a higher frequency than the coordinator ([1.83,3.32]). In case of SrcRI an opposite scenario is visible, where coordina-

<sup>4</sup>**Confidence Level** is a label which has been introduced by us, for measuring the confidence during the prediction of role stereotypes. The scale for the label is one to five, where five is the highest confidence level

tor (10.1) has a higher frequency over the controller (2.5). It could be we were biased when labeling the controller, as we have decided the labels based on the class names, method names and type of dependencies which are visible in UML class diagrams. Otherwise, it might be an error that occurred in the earlier case study, when labeling the classes based on the source code. As we are not sure about which study labelled them correctly, we looked at the combined occurrences of coordinator and controller in both study, then we see that there is not much difference.

By comparing between URI and SrcRI classifier, we can conclude that URI clas-



**Figure 4.4:** *Comparison between URI and SrcRI classifiers*

sifier is quite accurate during classifying the different role-stereotypes. So, we can make early predictions using URI classifier during software development process and in that way we can save time, cost and effort. An illustration of the comparison is illustrated in the figure 4.4.

#### 4.4.2.2 Machine Learning Perspective

In the previous part we have seen the differences between frequency of occurrences of role-stereotypes while classifying with URI and SrcRI. In this section we will discuss the differences between these two classifiers from the Machine Learning (ML) perspective. For this we have taken a confusion matrix from Fabian's case study [32], which is shown in the table 4.18, they produced this confusion matrix by running SrcRI (which is called CRI in Chaudron's paper [3]) classifier on their imbalanced dataset. For this reason, we have taken a confusion matrix from our experiment results, that we got from similar settings (see section 4.1.1). That means by running

URI (Random Forest) classifier on our imbalanced dataset, here we have considered All Confidence level cases. Our confusion matrix is illustrated in the table 4.19.

	IH	ST	SP	CT	CO	IT
IH	463	13	47	2	2	14
ST	21	24	41	1	0	13
SP	35	10	448	2	6	38
CT	5	1	19	29	1	8
CO	17	2	38	4	27	14
IT	10	3	42	1	2	144

	IH	ST	SP	CT	IF	CO
IH	158	0	9	0	0	0
ST	23	4	1	0	0	0
SP	39	0	63	0	1	0
CT	10	0	1	22	0	0
IF	26	1	6	0	14	0
CO	10	0	2	0	0	1

**Table 4.18:** Confusion Matrix for RF classifier for the imbalanced dataset (SrcRI)

**Table 4.19:** Confusion Matrix for RF classifier for the imbalanced dataset (URI)

From table 4.18, we can see that SrcRI has classified service provider (SP) and information holders (IH) more accurately than other role-stereotypes. From table 4.19, we can see that our classifier URI has done a similar job, it has classified SP and IH more accurately. SrcRI couldn't classify Controller (CT) classes properly and URI couldn't classify coordinator classes accurately. In case of SrcRI there are less controllers in the dataset, so it can be said that the classifier does not have many examples to train itself properly. In our case, we had few coordinator (CO) classes in our dataset, that's why the URI classifier could not classify CO classes accurately. This is actually a pretty common scenario when doing machine learning with imbalanced dataset. The SrcRI tool provides 74% to 98% classification accuracy [32], whereas our URI classifier provides 70% to 90% classification accuracy. From this discussion we can see that SrcRI and CRI has performed quite similarly.

## 4.5 Threats to validity

### Threats to internal validity:

It refers to the factors which jeopardizes internal validity. Internal validity refers to the cause and effect relationship between the experiment and the outcome. Lack of case studies related to our research can have a negative affect on the internal validity.

Besides, the size of the dataset might be increased, but manual labeling and other steps are time consuming, due to the time limit, increasing the data seemed to be a bit out of scope. The correctness of the ground truth can have an adverse affect on the internal validity. For reducing the experimenter bias, we have used the same dataset, same metrics or features for each experiment.

### Threats to external validity:

It refers to factors which threaten the external validity. External validity refers to how generalizable the findings of our case study. We have used the lindholmen dataset, source forge and the github as resources for our case study. It is difficult to get good selection of UML class diagrams. The UML class diagrams we have used

are from open source projects. These may not be representative for diagrams which are used in practice.

In this case study, we have considered only the .xmi files for our experiment, may be if we have used some other types of sources the results can be a bit different. This can be debated by saying, we didn't find that many projects with other sources, for instance, we have found images(.png, .jpg), but those images were not clear and there were insufficient tools for converting the images into concrete data. For avoiding this we have used the .xmi files, which were available than the other resources.



# 5

## Conclusion & Future Directions

In this chapter, we will draw a conclusion based on our findings and discussion. Besides, we will give a future outlook for the possible consecutive works for this research study.

### 5.1 Conclusions

In order to gain insight in the structure of software designs, we seek a way to systematically categorize the software projects in design level into stereotypes. From earlier studies in role-stereotypes in software, we know that there are stereo-typical responsibilities within a software and when used properly it leads to well-design systems.

In this research study we have presented an approach to label the software classes into stereotypes from UML classes; i.e. design level representation of system. After that, we have used several machine learning (ML) algorithms to compare and evaluate the labelling of role stereotypes. To be more specific, we manually labeled six types of roles for the class diagrams of 15 selected software projects. We have experimented with several ML algorithms. Now, we can see which labels are given correctly or more precisely in the manual labeling stage. Furthermore we can check which ML algorithm provides the best result in terms of classification accuracy. In the next two paragraphs, we will describe our steps of gathering the results in a concise manner.

The manual labelling result proves that it is possible to assign labels to software classes on the abstraction level provided by UML class diagrams. In our case the labeling of classes are done following Wirf Brock's categorization of role stereotype's [1]. However, this manual labeling process is preferably to be carried out within a well-defined background in order to guarantee the precision. Otherwise the outcome may come out with low precision rate which is not preferable. This step is to reduce the level of biases so we know we can have a commonly acknowledged result. In our projects, the authors have done the initial manual labelling. The complete labelling was reviewed by the supervisor Michel Chaudron and his PhD student Rodi Jolak (both of which are familiar with role stereotypes)

We have found that the J48 algorithm performs best when classifying the imbalanced dataset (without oversampling). It achieves an accuracy of 65.7 on the overall

dataset and 68.3 for the data in which the manual labellers have high-confidence. On the other hand, Random Forest (RF) performs best when classifying the entire data set with SMOTE oversampling (check 4.1): it achieves accuracy of 89.6. Random Forest did not show an improvement when focussing only on the high-confidence level (accuracy of 88.3). J48 comes close to this with an accuracy of 84.7.

Figure 4.3 shows that Random forest with SMOTE performs the best among all other machine learning algorithms. This figure 4.3 also shows how J48 performed better in case of imbalanced dataset. However, it is notable to say that even the classifier performs the best on an overall level, it does have different levels of performance for different stereotypes. For example, in table 4.2 it gives the best result on Information Holder, it gives medium good result on Service Provider, Interfacer and Controller, poor result on Structurer. Moreover, it failed to correctly classify any coordinators. One likely reason is the low number of occurrences of these roles in several projects. Thus there are few examples from which the ML-classifier can learn what are the key characteristics of this stereotype. Also it could be that there are other features which could be helpful in increasing the classification result but we did not take them into consideration. To address this, one can experiment with more feature extracting tools as well as perform model analyzing by ourselves to enhance the possibility of discovering new features. Other approaches such as increasing the size of the project, number of projects and selecting projects with more standardized class diagrams can also be considered in order to increase the accuracy for using classifier.

In Figure 3.6 we presented a pie chart with the overall distribution for all the role stereotypes. And we have discussed (in Section 4.4) that the distribution of the role stereotypes in this graph should be further analyzed and utilized in order to show the general distribution. As we have talked before, it is reasonable that if a project has a high number of IH and SP classes, and low number of CO and ST. It matches the description that Wirfs Brock[1] mentioned that a regular software project should spend most of its portions devoted to store and transfer some data [1]. And, in the meanwhile have a certain number of implementations that execute some tasks with the usage of those data. And then there should be a small amount of classes such as Interfacers (IF) that talk to another system or system layer and some Controller (CT) classes that make some sort of decisions among the system. Then we can conclude that if a software project's architecture shows a different type of distribution in terms of the role stereotypes then this project must serve a different purpose. Be it an utility-based project which will mostly have Service Provider (SP) classes.

To answer **RQ 1**, we can say that it is possible to classify role stereotypes for classes in UML diagrams. After we have manually labelled over 15 software projects, we can confirm that it is feasible to label classes in a software project based on human assessment. However, there are some preliminary requirements need to be fulfilled. It requires some extent of software knowledge, especially in the UML model area so that the evaluator or whoever is labelling the UML diagrams has some basic knowledge of the UML model and what purposes those diagrams serve. Then the evaluator should have a good understanding of the concept of role stereotypes. He

should also know the criteria to distinguish between one stereotype to another. Our labels reached a good level of agreement before we used it with machine learning algorithms.

For **RQ 2**, the features that we used to identify the role stereotypes for UML classes can be divided into two parts. We will describe them in the following two paragraphs.

### 5.1.1 Topological features

The first part is the automatically generated features that we derived from using tool SDMetrics. SDMetrics is well-known for classifying and extracting software features for a project. However, considering the format of UML classes are more often to appear in an image such as .png, .jpg and jpeg etc, and the SDMetrics tool is unable to directly process the UML diagrams in image format so we suggest to convert the UML diagrams from image format (.png, .jpg, .bmp etc) to XML format. The available ones are .XML, .XMI which are both accepted by SDMetrics. Some other less often used XML formats can worth a try in SDMetrics too but it is not guaranteed. A list of derived features can be viewed in section 3.2.3. So for the conclusion, our opinion is to get the basic metrics such as NumAttr, NumOps for a UML diagram and use it as a foundation to classify the role stereotypes. And not all of the features are necessary in order to classify the stereotypes, the irrelevant ones should be tested and removed. For that part, we explained our process of filtering out the irrelevant features in section 3.2.3 too.

### 5.1.2 Features based on class-names

In table4.16 there are features named in a different way: EndWithManager, EndWithFactory/Impl/Implementation (EndWithFact) etc. These features are made by the researcher after analyzing the software project as well as the UML class diagrams for that project. In this list we have ranked the importance of each feature and how they affected our automated classification process using machine learning. From the table we can see that most of the derived features has a high importance value except for the EndWithManager/Controller (EndWithMan) feature. This feature was part of the criteria which we used to determine the Controller (CT) role stereotype within a project. So what we can conclude in this table, is that most of the naturally derived features such as OperationInteritance (OpsInh), NumAttr and NumOps should be fine to be considered as the first hand metrics to classify role stereotypes. Especially the features provided by SDMetrics, it has given reliable result and high accuracy in terms of automated role labeling. However, if it is preferred to increase the confidence level of the result, it is acceptable to add own metrics which can be useful in most of the software projects. In our case, the recommended customized features are EndWithManager, HasType/Annotation/List and EndWithFactory as they can be used to detect multiple stereotypes and also has a good ranking result.

For answering **RQ 3**, we need to look at section 4.1.1 and section 4.1.2. In those section, we have analyzed the accuracy table and confusion matrix from our experiments. We have 2 datasets: one with All Confidence Level <sup>4</sup>. Another dataset contains only datapoints with High Confidence Level. In each experiment, we have 2 segments, in first segment we run our classifier URI (UML Role Identifier) for the imbalance dataset, and in the the second segment we have run URI on the dataset after applying SMOTE resampling technique. For getting an overview of how different machine learning algorithms performed for our 2 dataset, look at table 4.3. After analyzing all these results, we have found that J48 machine learning (ML) algorithm performs best when classifying the imbalance dataset and Random Forest ML algorithm performs best when classifying the dataset with SMOTE. An illustration of these classifiers performances can be observed from this figure 4.3.

### 5.2 Future Directions

This research study can serve as a basis for follow-up studies.

In our case study, we have focused only on one type of UML models, which is UML class diagrams. There is another type of UML diagram which might be interesting to work with for a similar kind of research. The UML diagram type we are talking about is sequence Diagram. This study can be an extension of our research, which will go through similar kind of experiment, but with sequence diagram. Later, a comparison can be made between our result and the outcome from this future study. We have shown some examples of sequence diagram from the K9 projects in the appendix D.

Another continuation of our case study can be running the same experiment with a larger data set. Currently, we have developed a ground truth of 391 classes from 15 projects (see 3.2.5.3). The main challenging part of the future work will be to gather relevant projects and labeling the classes manually. For our case study, the whole process took about 3 months. By doing an extension of our study, the classification accuracy of the machine learning algorithms might be improved. As more example instances means more data to train the classifier.

Our case study has contributed by providing a classifier which can classify the role-stereotypes in the UML class diagrams. Another contribution of this research is that, all the datasets and analysis are made available through a github repository <sup>5</sup>. There are many other types of UML models which can be used for similar types of research, we kept these options open for future work.

---

<sup>4</sup>**Confidence Level** is a label which has been introduced by us, for measuring the confidence during the prediction of role stereotypes. The scale for the label is one to five, where five is the highest confidence level

<sup>5</sup><https://github.com/hammer007/umlRoleIdentifier>

# Bibliography

- [1] WirfsBrock, R. J. (2006). Characterizing classes. *IEEE software*, 23(2), 911.
- [2] Thung, F., Lo, D., Osman, M. H., Chaudron, M. R. (2014, June). 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).
- [3] Nurwidyantoro, A., HoQuang, T., & Chaudron, M.R.V. (2019, April). Automated classification of class role stereotypes via machine learning. In *Proceedings of the Evaluation and Assessment on Software Engineering* (pp. 7988). ACM.
- [4] Osman, Mohd Hafeez, Michel R.V. Chaudron, and Peter Van Der Putten. An analysis of machine learning algorithms for condensing reverse engineered class diagrams. In *2013 IEEE International Conference on Software Maintenance*, pp. 140-149. IEEE, 2013.
- [5] Baltes, S., Diehl, S. (2014, November). Sketches and diagrams in practice. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 530-541). ACM.
- [6] Alhindawi, N., Dragan, N., Collard, M. L., Maletic, J. I. (2013, September). Improving feature location by enhancing source code with stereotypes. In *2013 IEEE International Conference on Software Maintenance* (pp. 300309). Ieee.
- [7] Dragan, N., Collard, M. L., Maletic, J. I. (2010, September). Automatic identification of class stereotypes. In *2010 IEEE International Conference on Software Maintenance* (pp. 110). IEEE.
- [8] Moreno, L., Marcus, A. (2012, September). Jstereocode: automatically identifying method and class stereotypes in java code. In *Proceedings of the 27th IEEEACM International Conference on Automated Software Engineering* (pp. 358361). ACM.
- [9] HoQuang, T., Chaudron, M. R. V., Samúelsson, I., Hjaltason, J., Karasneh, B., Osman, H. (2014, December). Automatic classification of UML class diagrams from images. In *2014 21st AsiaPacific Software Engineering Conference* (Vol.

- 1, pp. 399406). IEEE.
- [10] Moha, N., Gueheneuc, Y. G., Leduc, P. (2006, September). Automatic generation of detection algorithms for design defects. In 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06) (pp. 297300). IEEE.
- [11] Ali, N., Lai, R. (2014, November). Managing requirements change in global software development. In 2014 International Conference on Data and Software Engineering (ICODSE) (pp. 15). IEEE.
- [12] Debnath, N. C., Uzal, R., Montejano, G., Riesco, D. (2006, May). Software projects leadership: elements to redefine" risk management" scope and meaning. In 2006 IEEE International Conference on ElectroInformation Technology (pp. 280284). IEEE.
- [13] Hebig, R., Quang, T. H., Chaudron, M. R. V., Robles, G., Fernandez, M. A. (2016, October). The quest for open source projects that use UML: mining GitHub. In Proceedings of the ACMIEEE 19th International Conference on Model Driven Engineering Languages and Systems (pp. 173183). ACM.
- [14] Witten, I. H., Frank, E., Hall, M. A., Pal, C. J. (2016). Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann.
- [15] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I. H. (2009). The WEKA data mining software: an update. ACM SIGKDD explorations newsletter, 11(1), 1018.
- [16] Vinco, S. (2019, September 23). Random forest. Retrieved October 15, 2019, from [https://en.wikipedia.org/wiki/Random\\_forest/media/File:Kernel\\_Machine.svg](https://en.wikipedia.org/wiki/Random_forest/media/File:Kernel_Machine.svg).
- [17] Hebig, R., Quang, T. H., Chaudron, M. R. V., Robles, G., Fernandez, M. A. (2016, October). The quest for open source projects that use UML: mining GitHub. In Proceedings of the ACMIEEE 19th International Conference on Model Driven Engineering Languages and Systems (pp. 173183). ACM.
- [18] Budgen, D. (2003). Software design. Pearson Education.
- [19] Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., Young, P. R. (1989). Computing as a discipline. Computer, 22(2), pp6370.
- [20] Jones, J. C. (1992). Design methods. John Wiley Sons.
- [21] M.R.V. Chaudron, W. Heijstek, A. Nugroho (2012). How effective is UML modelling? An empirical perspective on costs and benefits. Softw Syst Model,

pp571580.

- [22] Seidewitz, E. (2003). What models mean. *IEEE software*, 20(5), 26-32.
- [23] Genero, M., Piattini, M., Calero, C. (2005). A survey of metrics for UML class diagrams. *Journal of object technology*, 4(9), pp5992.
- [24] Kotsiantis, S. B., Zaharakis, I., Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160, pp324.
- [25] Alghamdi, M., AlMallah, M., Keteyian, S., Brawner, C., Ehrman, J., Sakr, S. (2017). Predicting diabetes mellitus using SMOTE and ensemble machine learning approach: The Henry Ford Exercise Testing (FIT) project. *PloS one*, 12(7).
- [26] Brownlee, J. (2019). *Machine learning mastery with Weka*. Ebook. Edition: v. 1.4.
- [27] Chawla, N. V., Bowyer, K. W., Hall, L. O., Kegelmeyer, W. P. (2002). SMOTE: synthetic minority oversampling technique. *Journal of artificial intelligence research*, 16, 321357.
- [28] Padhi, S., Millstein, T., Nori, A., Sharma, R. (2019, July). Overfitting in synthesis: Theory and practice. In *International Conference on Computer Aided Verification* (pp. 315-334). Springer, Cham.
- [29] Buddhinath, G., Derry, D. (2006). A simple enhancement to one rule classification. *Department of Computer Science Software Engineering. University of Melbourne, Australia*.
- [30] Etxeberria, L., Sagardui, G. (2008, May). Quality assessment in software product lines. In *International Conference on Software Reuse* (pp. 178181). Springer, Berlin, Heidelberg.
- [31] WanSeob, B., Bo, Y., SaKyun, J., OkBae C. (2004). Extension and implementation of iconic stereotype for GNSS application in the UML class diagram. *2004 International Conference on Cyberworlds* (pp. 162169). Tokyo, Japan.
- [32] Fabian Fröding, Duy Nguyen Ngoc (2020). *The Evolution of Role-Stereotypes and Related Design (Anti)Patterns*. B.Sc. thesis Chalmers & Gothenburg University, Department of Computer Science and Engineering.
- [33] A. Nugroho, M. R. V. Chaudron and E. Arisholm, Assessing UML design metrics for predicting fault-prone classes in a Java system, 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), Cape

Town, 2010, pp. 21-30.

- [34] Ana M. Fernández-Sáez, Michel R. V. Chaudron, Marcela Genero, and Isabel Ramos. 2013. Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance? a controlled experiment. In Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE '13). Association for Computing Machinery, New York, NY, USA, 60–71. DOI:<https://doi.org/10.1145/2460999.2461008>.
- [35] Fernández-Sáez, A.M., Genero, M., Caivano, D. et al. Does the level of detail of UML diagrams affect the maintainability of source code?: a family of experiments. *Empir Software Eng* 21, 212–259 (2016). <https://doi.org/10.1007/s10664-014-9354-4>.





# A

## Appendix A

Sequence	Name of the Project	UML file type
1.	ACSUFRO	.xmi
2.	Bioclipse_brunn	.xmi
3.	Bitys	.xmi
4.	Calligra	.xmi
5.	Object_Course_End	.xmi
6.	Green_House_Xml_Parser	.xmi
7.	SE_Project	.xmi
8.	Talon	.xmi
9.	Pizza_Delivery_System	.xmi
10.	FIB-PROP	.png
11.	example-cloudbigdata-app	.png
12.	ModuloCaixaSugestao	.png
13.	JavaChatAgain	.xmi
14.	agendaidt	.png
15.	BigPrototype15	.jpg
16.	CandyCrush	.xmi
17.	ClimateControlSytem	.uml
18.	html5multi	.gif
19.	JTravelAgency	.jpg
20.	QtOpenDynoScreen	.xmi
21.	Mvp4g	.png
22.	Oodp_lab	.jpg
23.	TravelDream	.svg
24.	Visuwall	.jpg
25.	JCoAP	.xmi
26.	Games	.xmi
27.	JGAP	.xml
28.	Wro4j	.xml
29.	Xuml	.xml
30.	JavaClient	.xml
31.	Neuroph	.xmi
32.	Mars-Simulation	.xml
33.	JPMC	.xml

**Table A.1:** List of Projects

Seq.	Repository Link
1.	<a href="https://github.com/jbarriosc/ACSUFRO/blob/master/LGPL/CommonSoftware/acsGUIs/alarmsDefGUI/doc/class_diagram.xmi">https://github.com/jbarriosc/ACSUFRO/blob/master/LGPL/CommonSoftware/acsGUIs/alarmsDefGUI/doc/class_diagram.xmi</a>
2.	<a href="https://github.com/jonalv/bioclipse.brunn/blob/master/docs/classDiagram.xmi">https://github.com/jonalv/bioclipse.brunn/blob/master/docs/classDiagram.xmi</a>
3.	<a href="https://www.github.com/shnee/bitys/tree/master/doc/class_diagram.xmi">https://www.github.com/shnee/bitys/tree/master/doc/class_diagram.xmi</a>
4.	<a href="https://www.github.com/wyuka/calligra/tree/highlighter/kexi/migration/xbase/doc/Class Diagram.png">https://www.github.com/wyuka/calligra/tree/highlighter/kexi/migration/xbase/doc/Class Diagram.png</a>
5.	<a href="https://github.com/Jackerty/ObjectCouresEnd/blob/master/doc/ClassDiagram.xmi">https://github.com/Jackerty/ObjectCouresEnd/blob/master/doc/ClassDiagram.xmi</a>
6.	<a href="https://www.github.com/JackKarichkovskiy/greenhouse-xml-parser/tree/master/diagram/classDiagram.xmi">https://www.github.com/JackKarichkovskiy/greenhouse-xml-parser/tree/master/diagram/classDiagram.xmi</a>
7.	<a href="https://www.github.com/a7medfahmy94/SE_Project/tree/master/Phase 2 extended/SocialNetworkClassDiagram.jpg">https://www.github.com/a7medfahmy94/SE_Project/tree/master/Phase 2 extended/SocialNetworkClassDiagram.jpg</a>
8.	<a href="https://www.github.com/929528/talon/tree/master/talon.xmi">https://www.github.com/929528/talon/tree/master/talon.xmi</a>
9.	<a href="https://github.com/TCameron/PizzaDeliverySystem/blob/master/trunk/docs/class_diagram.xmi">https://github.com/TCameron/PizzaDeliverySystem/blob/master/trunk/docs/class_diagram.xmi</a>
10.	<a href="https://github.com/jmigual/FIB-PROP/blob/master/EntregLatex/Diagrama3.png">https://github.com/jmigual/FIB-PROP/blob/master/EntregLatex/Diagrama3.png</a>
11.	<a href="https://github.com/ZuInnoTe/example-cloudbigdata-app/blob/master/doc/img/architecture/designarchitecture.png">https://github.com/ZuInnoTe/example-cloudbigdata-app/blob/master/doc/img/architecture/designarchitecture.png</a>
12.	<a href="https://github.com/zucoloto/ModuloCaixaSugestao/blob/master/modelagem/Diagrama%20de%20Classe/Diagrama%20de%20Classe.png">https://github.com/zucoloto/ModuloCaixaSugestao/blob/master/modelagem/Diagrama%20de%20Classe/Diagrama%20de%20Classe.png</a>
13.	<a href="https://www.github.com/91kerezi/JavaChatAgain/tree/master/spec/server.xmi">https://www.github.com/91kerezi/JavaChatAgain/tree/master/spec/server.xmi</a>

**Table A.2:** Repository Link of Projects from Table A.1

Seq.	Repository Link
14.	<a href="https://www.github.com/acamiestudios/agendaidt/tree/master/protected/modules/cruge/doc/Diagrama-de-clases-proceso-de-autenticacion2.png">https://www.github.com/acamiestudios/agendaidt/tree/master/protected/modules/cruge/doc/Diagrama-de-clases-proceso-de-autenticacion2.png</a>
15.	<a href="https://github.com/joprecht/bigPrototype15/blob/master/prototyp%20videoshop/app/src/main/analysis/eCateringClassDiagram.jpg">https://github.com/joprecht/bigPrototype15/blob/master/prototyp%20videoshop/app/src/main/analysis/eCateringClassDiagram.jpg</a>
16.	<a href="https://www.github.com/seddikouiss/candycrush/tree/master/diagramme_classe/MD/diagramme_apres_TP_Noter.xmi.xmi">https://www.github.com/seddikouiss/candycrush/tree/master/diagramme_classe/MD/diagramme_apres_TP_Noter.xmi.xmi</a>
17.	<a href="https://www.github.com/MykytaPonikarov/Test/tree/master/ClimateControlSystem/diagram/ControllerDetailsController.uml">https://www.github.com/MykytaPonikarov/Test/tree/master/ClimateControlSystem/diagram/ControllerDetailsController.uml</a>
18.	<a href="https://www.github.com/ziarrek/html5-multi/tree/master/server/serverGame/UMLServer.jpg">https://www.github.com/ziarrek/html5-multi/tree/master/server/serverGame/UMLServer.jpg</a>
19.	<a href="https://github.com/mvaraga/JTravelAgency/blob/master/Class%20Diagram1.jpg">https://github.com/mvaraga/JTravelAgency/blob/master/Class%20Diagram1.jpg</a>
20.	<a href="https://www.github.com/fville/QtOpenDynoScreen/tree/master/source/AnalogWidgets/Doc/AnalogWidgets.xmi">https://www.github.com/fville/QtOpenDynoScreen/tree/master/source/AnalogWidgets/Doc/AnalogWidgets.xmi</a>
21.	<a href="https://github.com/mvp4g/mvp4g/blob/master/etc/uml/mvp4g_class_diagram_overview.png">https://github.com/mvp4g/mvp4g/blob/master/etc/uml/mvp4g_class_diagram_overview.png</a>
22.	<a href="https://github.com/zubidlo/oodp_labs/blob/master/src/lab_7/lab_5.part_2_ULM_class_diagram.jpg">https://github.com/zubidlo/oodp_labs/blob/master/src/lab_7/lab_5.part_2_ULM_class_diagram.jpg</a>
23.	<a href="https://www.github.com/teopalva/travel-dream/tree/master/Deliveries/Documentation/ClassdiagramRASD.svg">https://www.github.com/teopalva/travel-dream/tree/master/Deliveries/Documentation/ClassdiagramRASD.svg</a>
24.	<a href="https://www.github.com/n0rad/visuwall/tree/master/projects-for-testing/struts/core/src/main/java/org/apache/struts/validator/doc-files/validatorUML.jpg">https://www.github.com/n0rad/visuwall/tree/master/projects-for-testing/struts/core/src/main/java/org/apache/struts/validator/doc-files/validatorUML.jpg</a>
25.	<a href="https://github.com/dapaulid/JCoAP/blob/master/Documentation/ClassDiagram/JCoAP.xmi">https://github.com/dapaulid/JCoAP/blob/master/Documentation/ClassDiagram/JCoAP.xmi</a>
26.	<a href="https://github.com/dgschwind/Games/blob/master/src/org/douggschwind/games/boardgames/monopoly/ClassDiagrams.xmi">https://github.com/dgschwind/Games/blob/master/src/org/douggschwind/games/boardgames/monopoly/ClassDiagrams.xmi</a>

**Table A.3:** Repository Link of Projects from Table A.1

# B

## Appendix B

In this appendix, we have illustrated the colored UML class diagrams of the 15 projects that we have created for the evaluation.

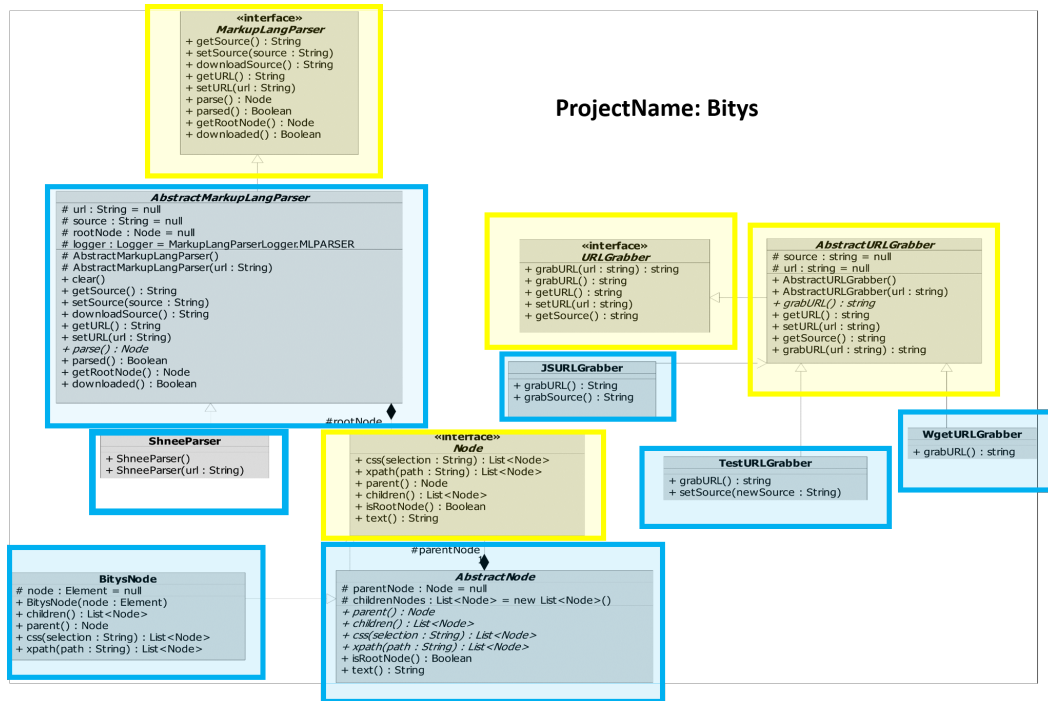


Figure B.1: *Bitys* UML class Diagram

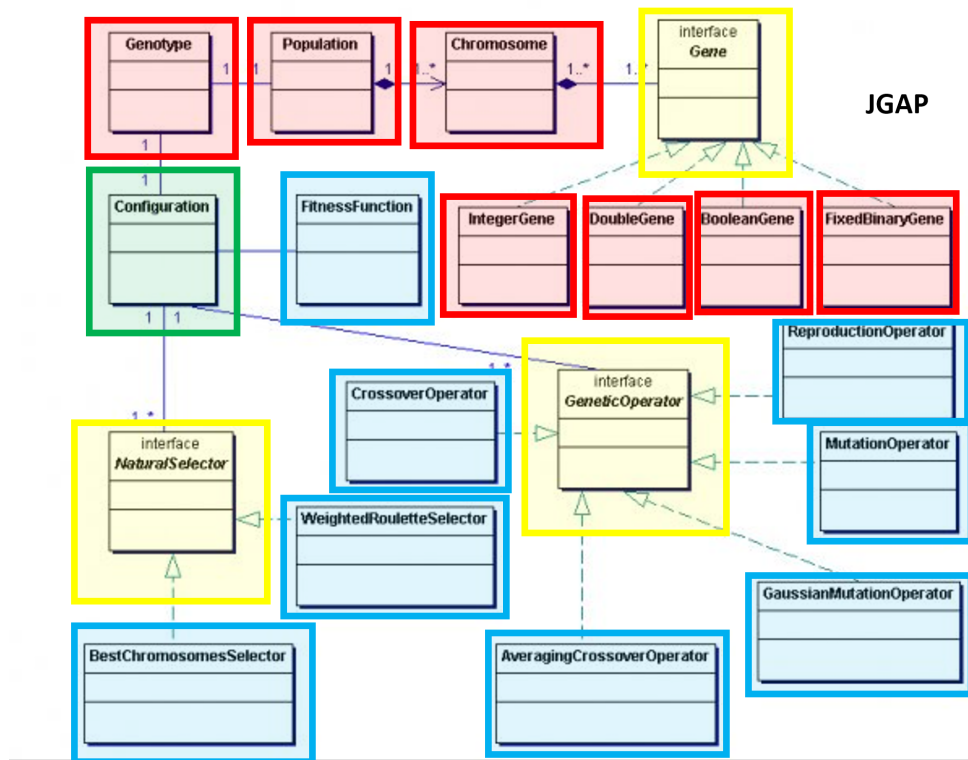


Figure B.2: JGAP UML class Diagram

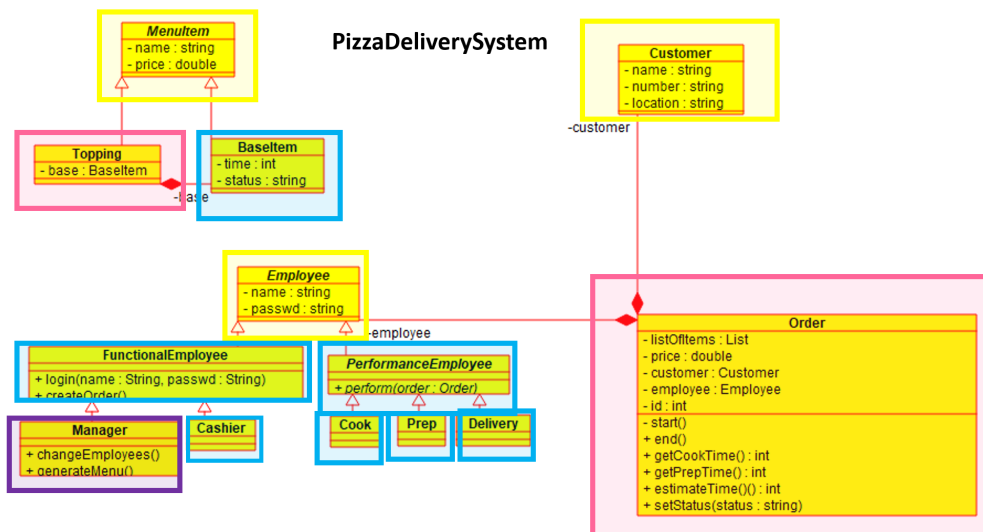
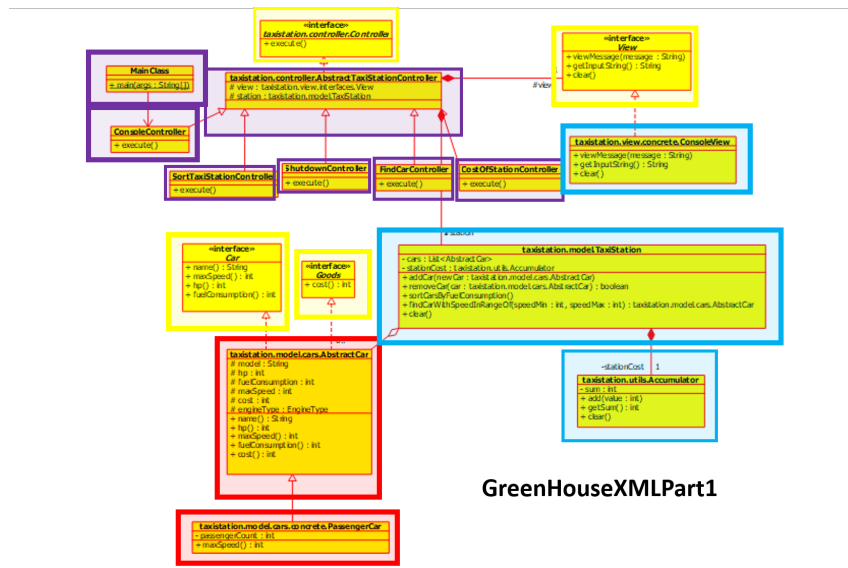
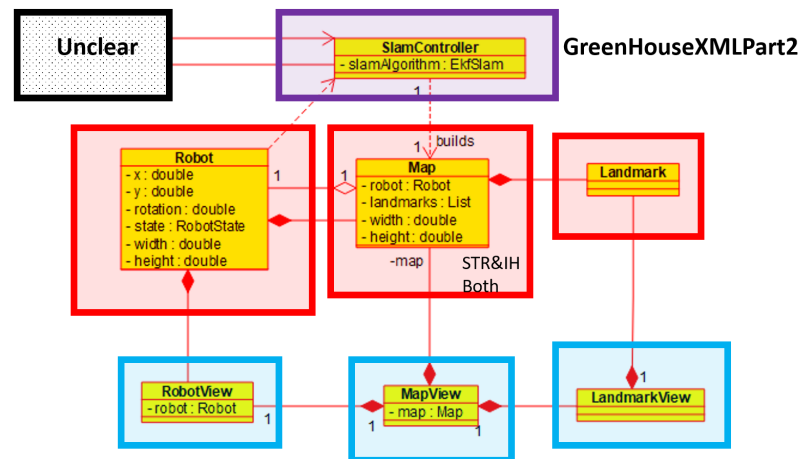


Figure B.3: Pizza Delivery System UML class Diagram

Figure B.4: *GreenHouseXmlParser* UML class Diagram (part 1)Figure B.5: *GreenHouseXmlParser* UML class Diagram (part 2)

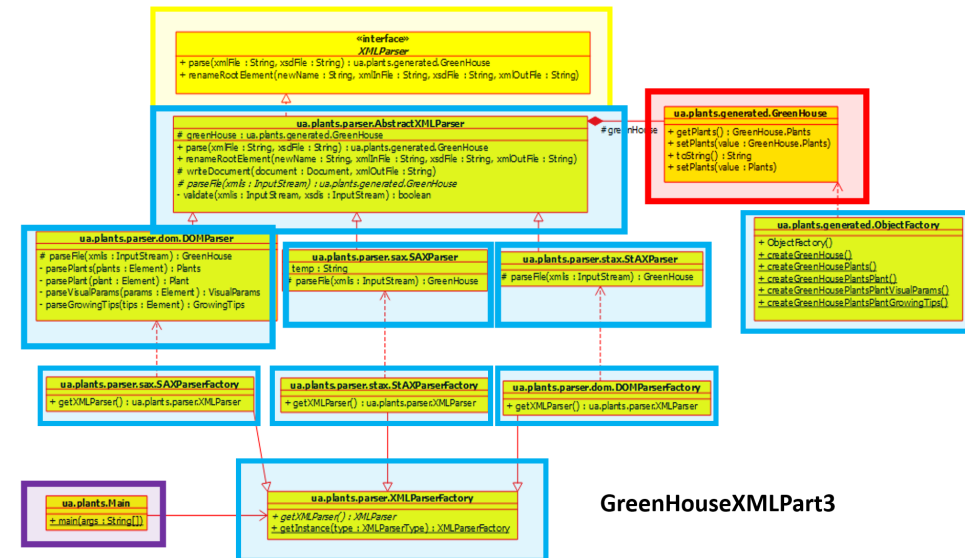


Figure B.6: *GreenHouseXmlParser* UML class Diagram (part 3)

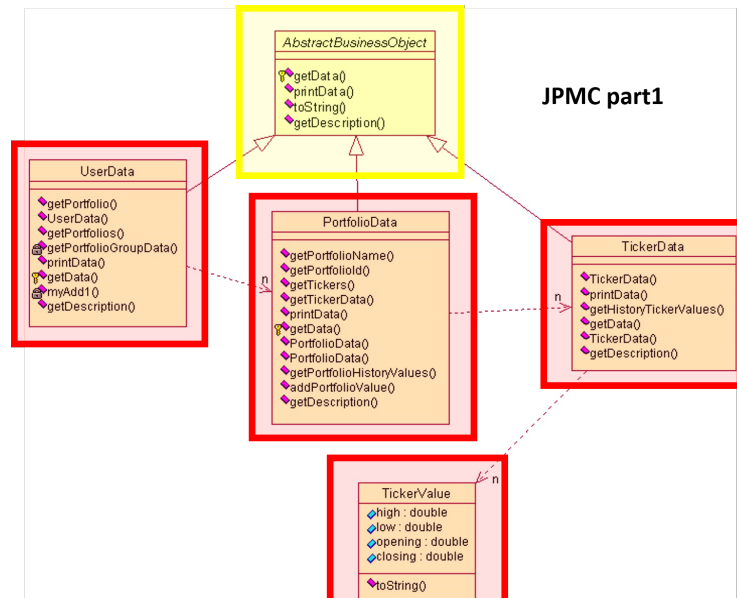


Figure B.7: *JPMC* UML class Diagram (part 1)



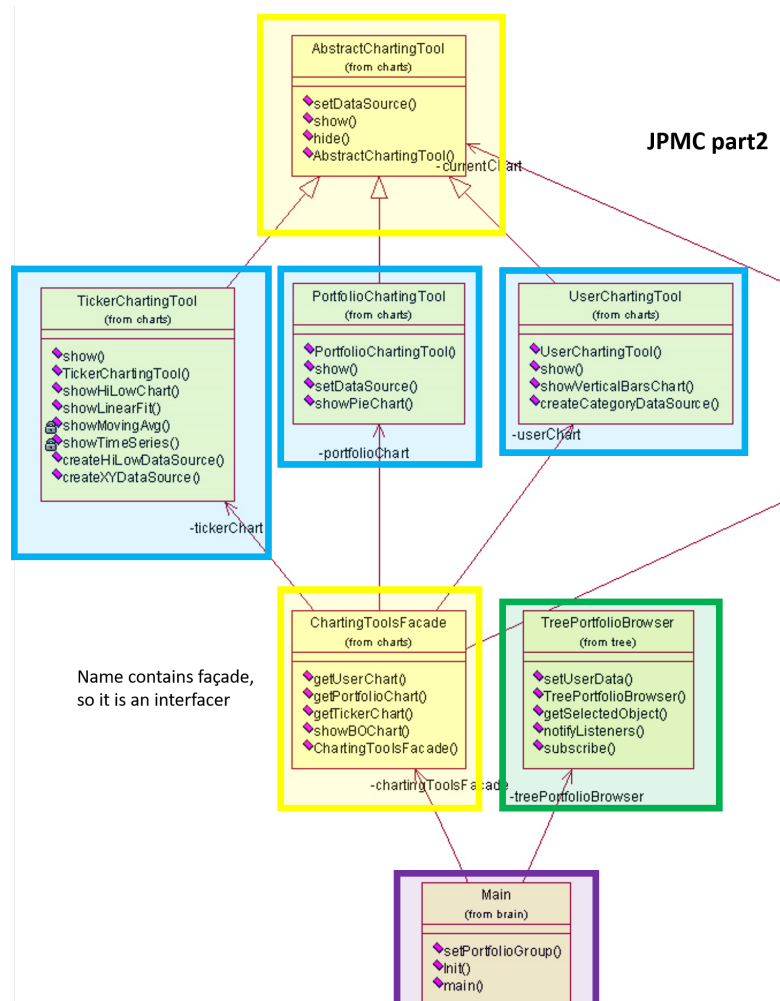


Figure B.8: JPMC UML class Diagram (part 2)

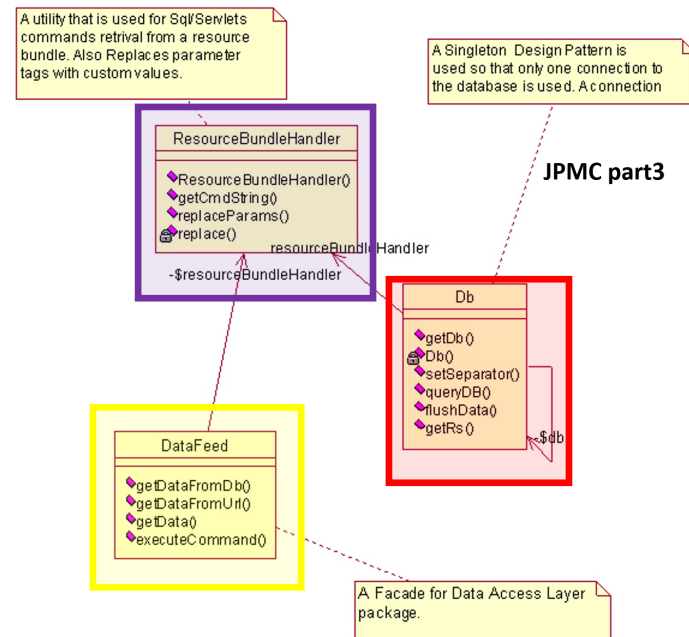


Figure B.9: JPMC UML class Diagram (part 3)

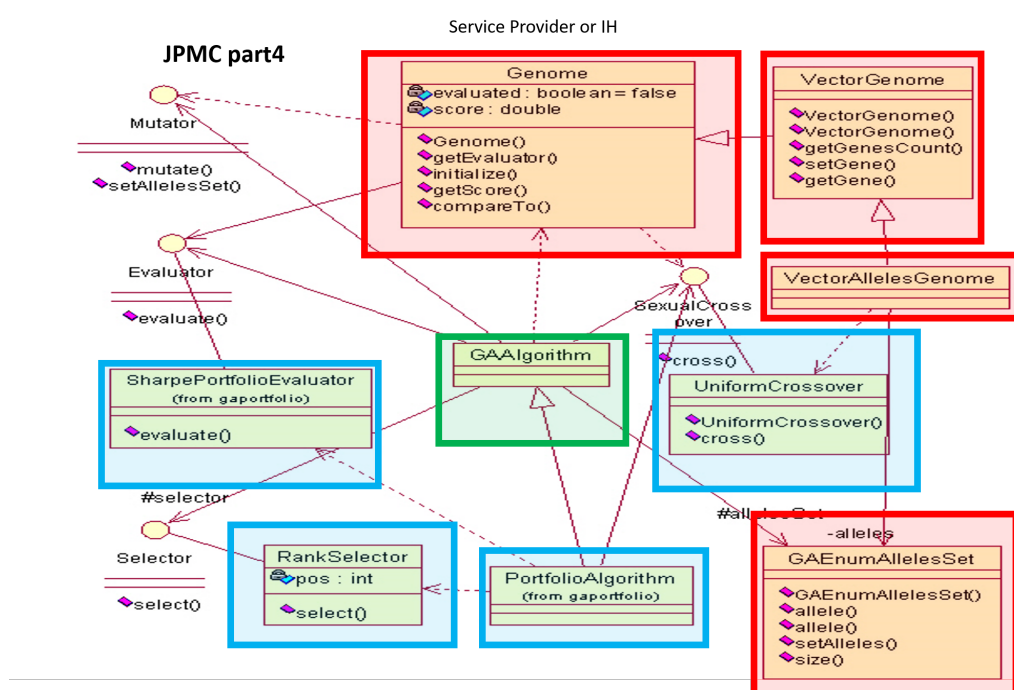
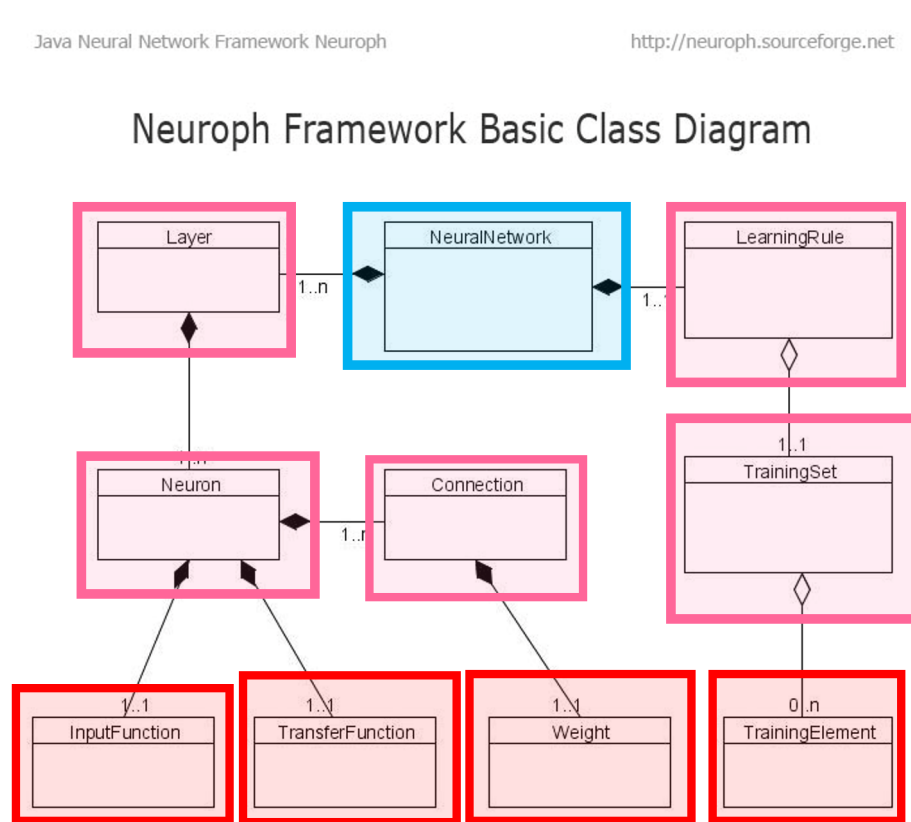


Figure B.10: JPMC UML class Diagram (part 4)



**Figure B.11:** *Neuroph UML class Diagram (part 1)*

## Neuroph Framework 2.3

### Core Class Diagram

Java Neural Network Framework  
<http://neuroph.sourceforge.net>

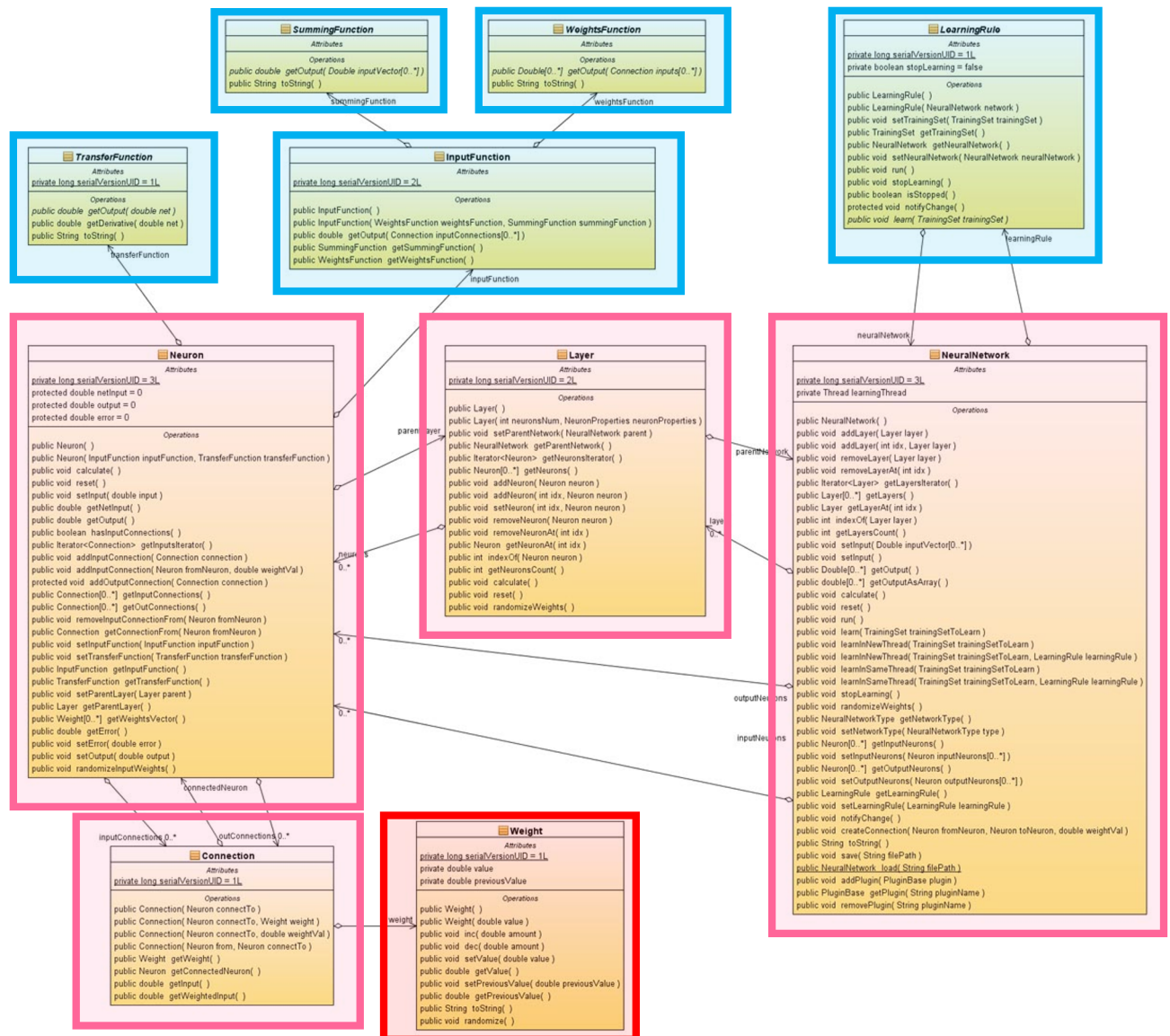


Figure B.12: Neuroph UML class Diagram (part 2)

## Neuroph Framework 2.3

### Learning Rules Class Diagram

Java Neural Network Framework  
<http://neuroph.sourceforge.net>

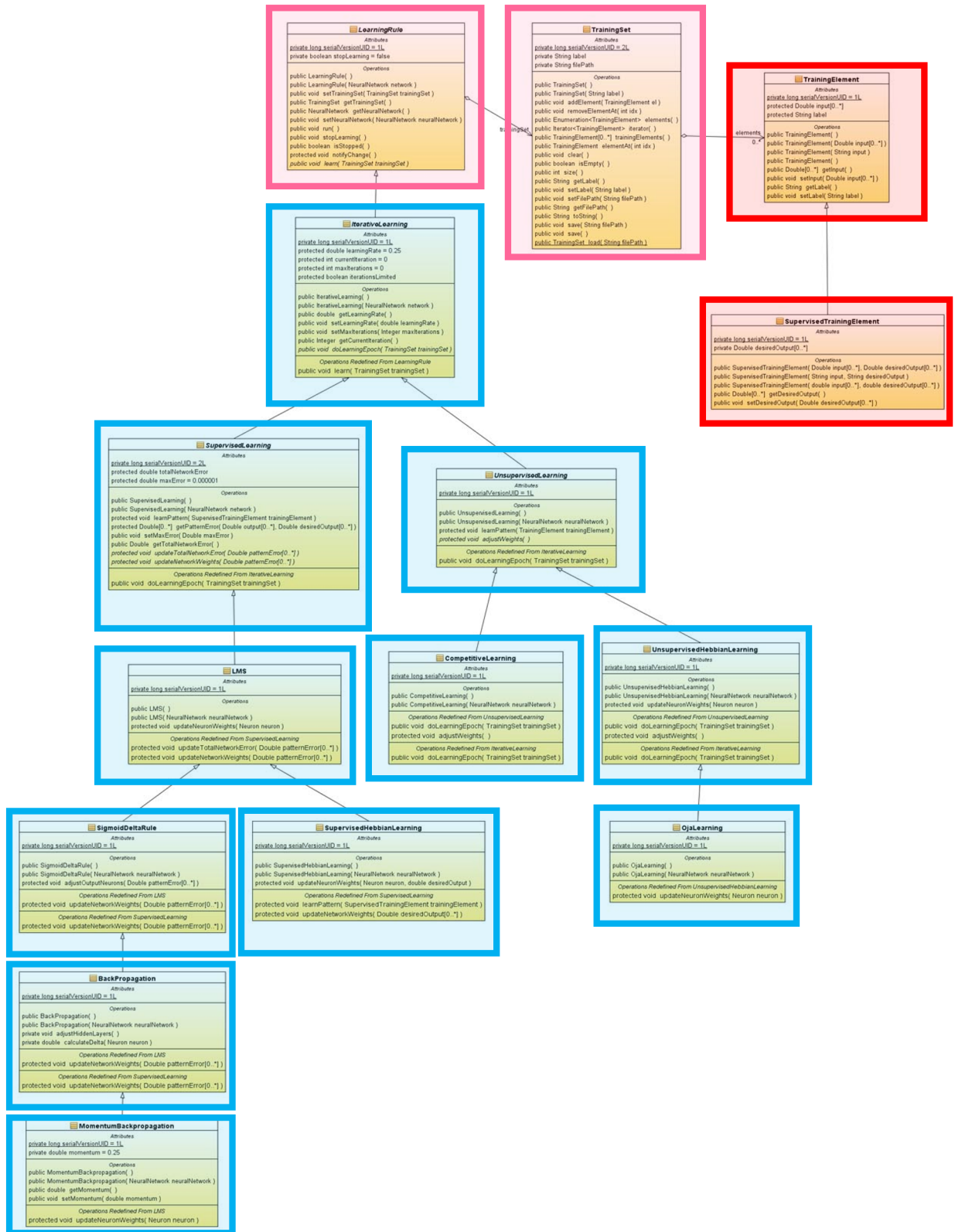
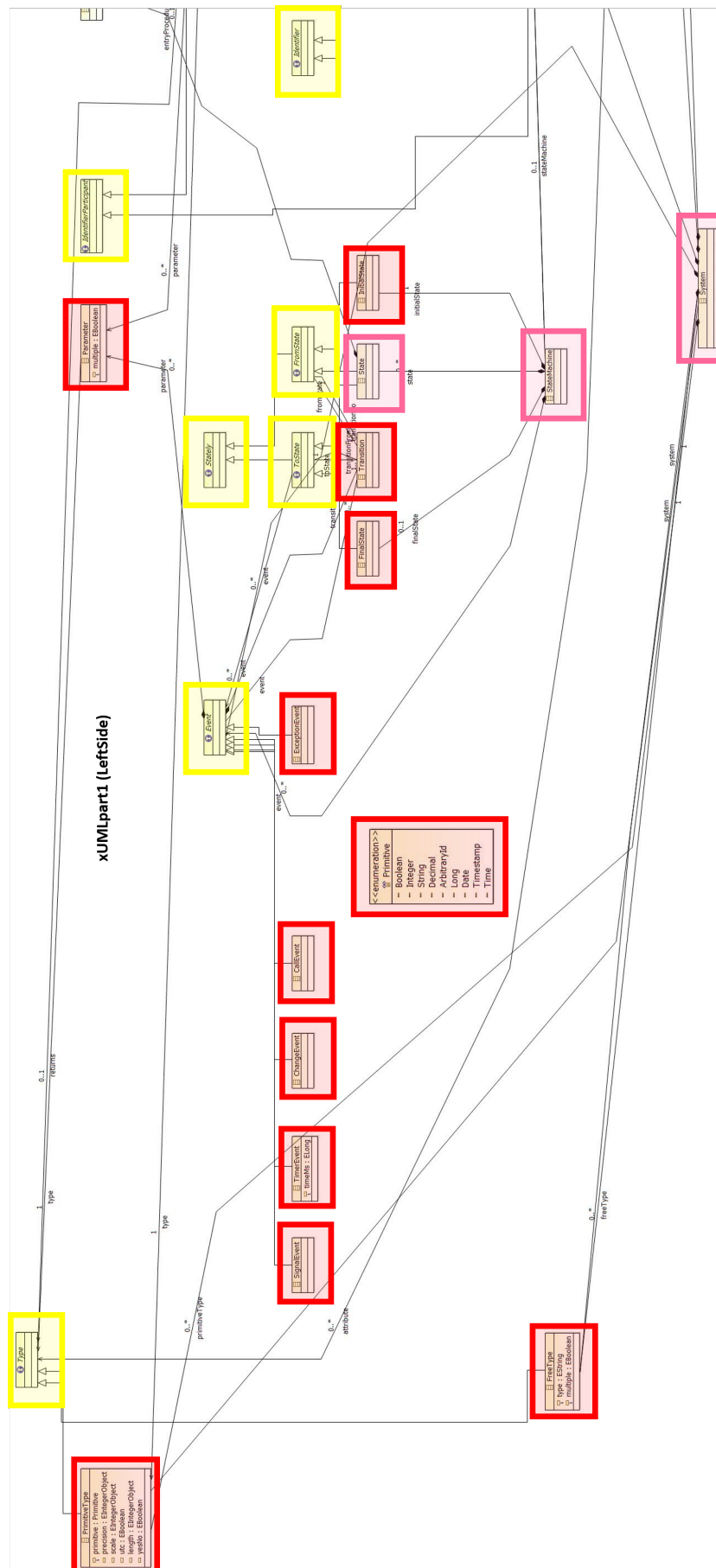


Figure B.13: Neuroph UML class Diagram (part 3)







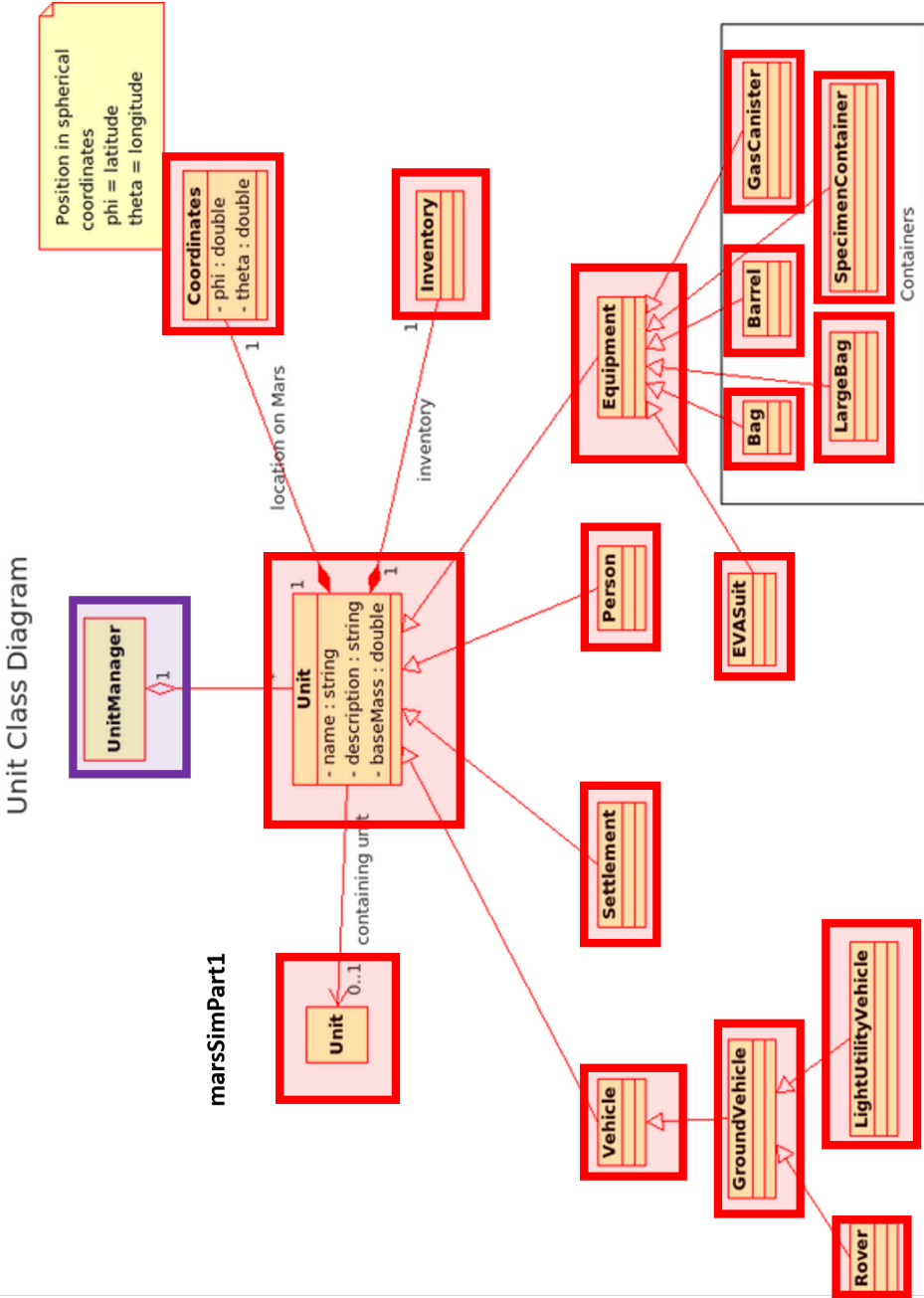
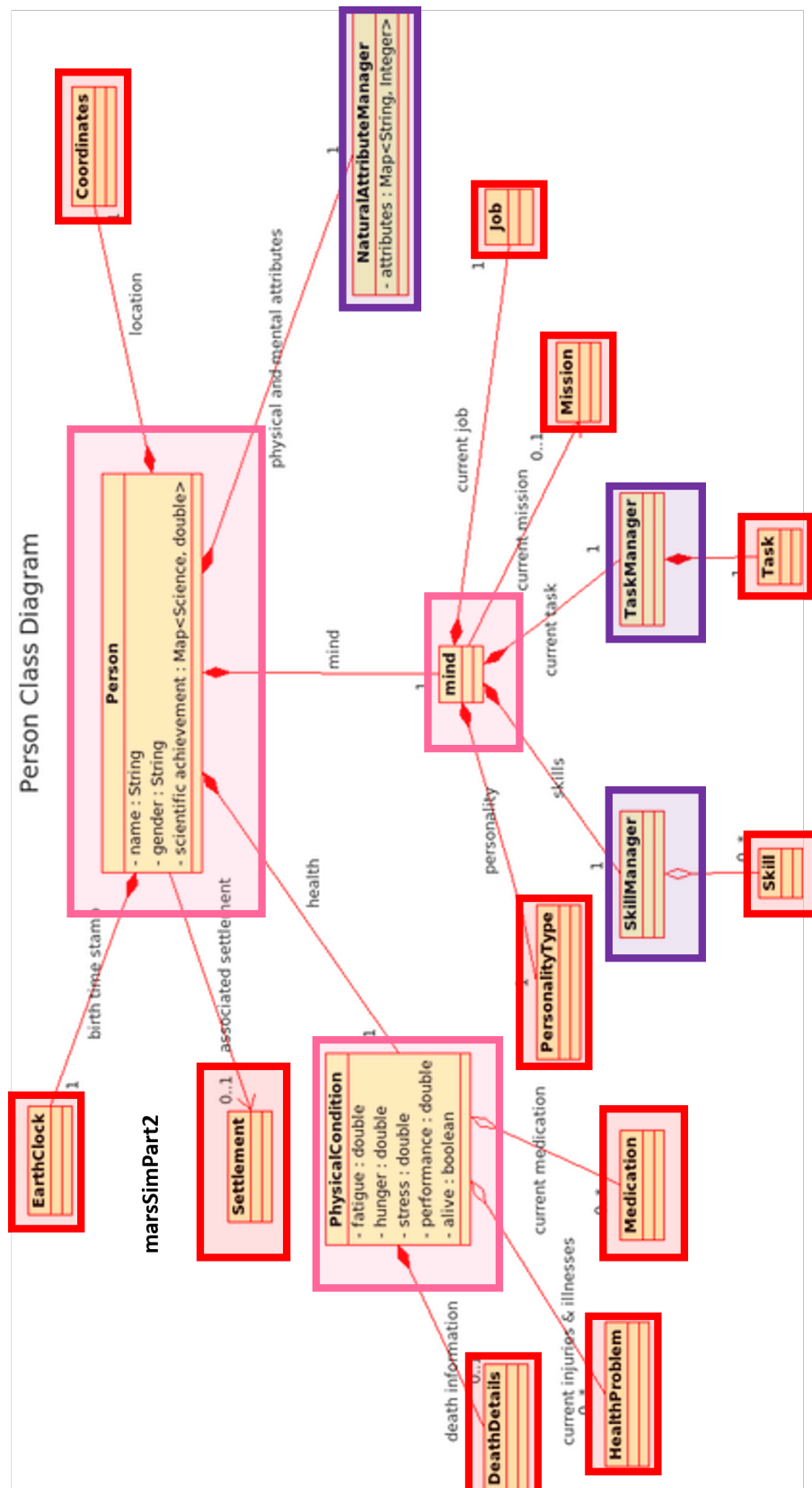
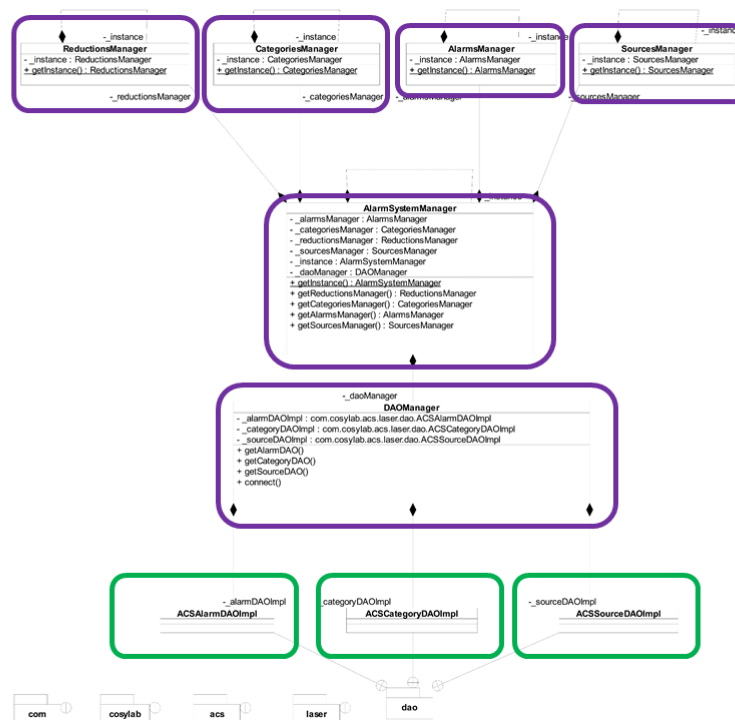


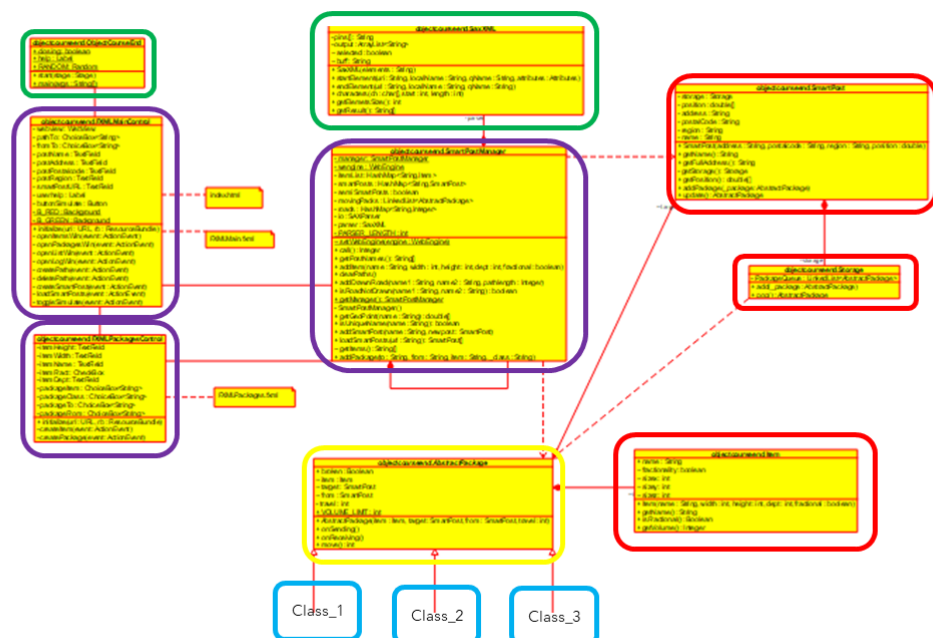
Figure B.16: *MarsSimulation UML class Diagram (part 1)*



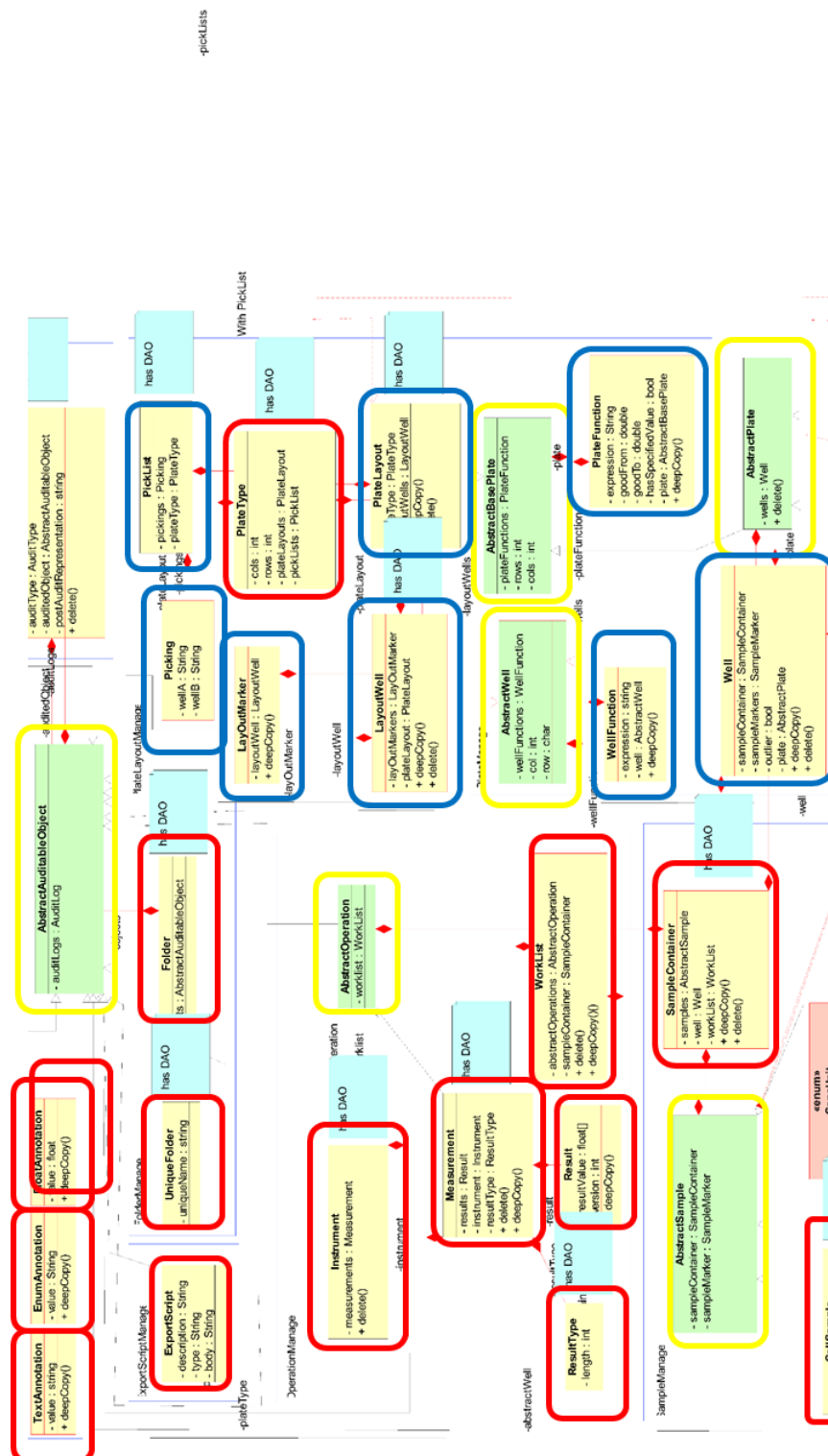
Figure B.17: *MarsSimulation* UML class Diagram (part 2)



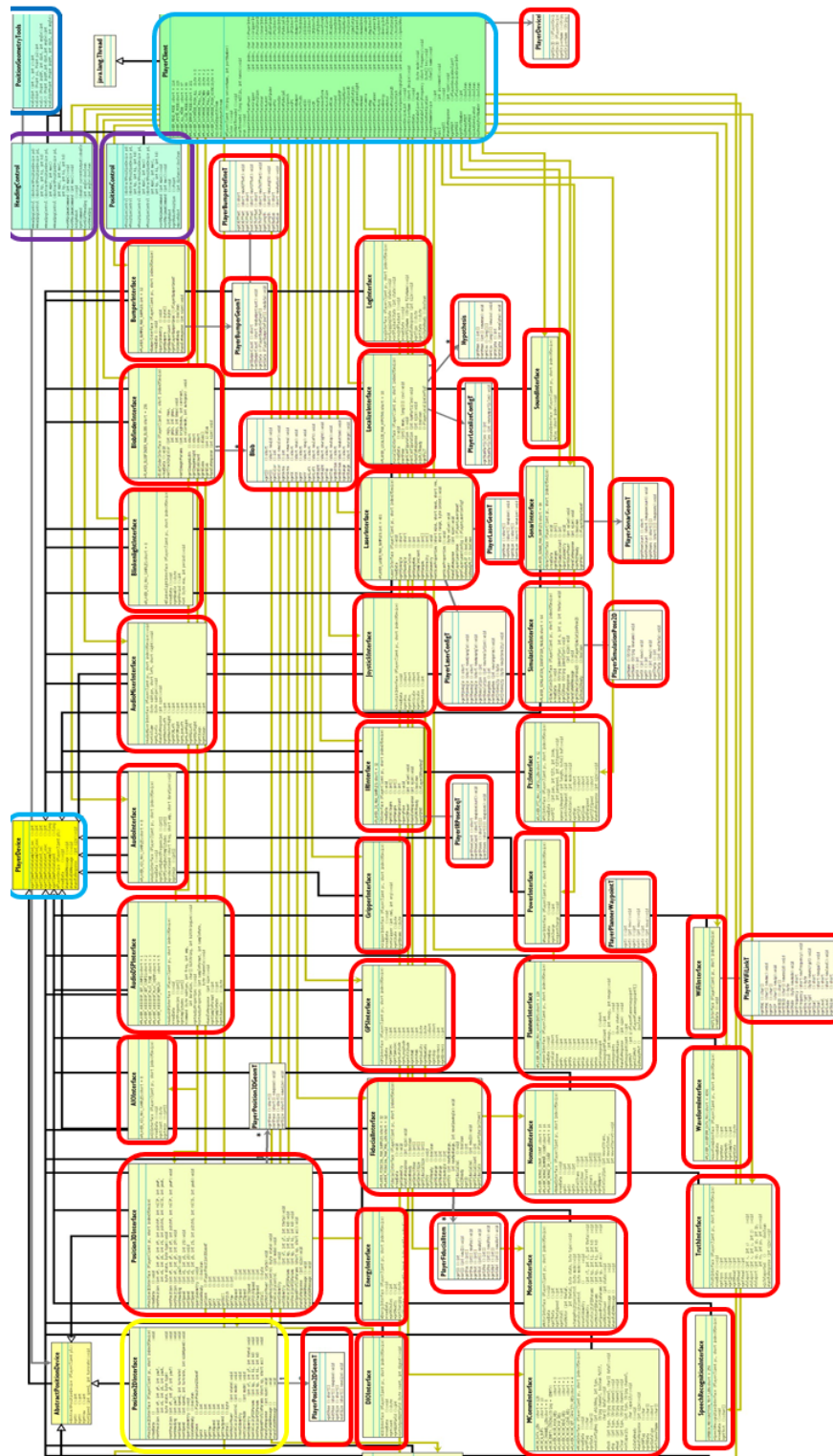
**Figure B.18:** *ACSUFRO UML class Diagram*



**Figure B.19:** *ObjectCourseEnd UML class Diagram*



**Figure B.20:** *BioclipseBrunn UML class Diagram*



**Figure B.21:** *Java\_client UML class Diagram*

**Figure B.22:** *SE\_project UML class Diagram*

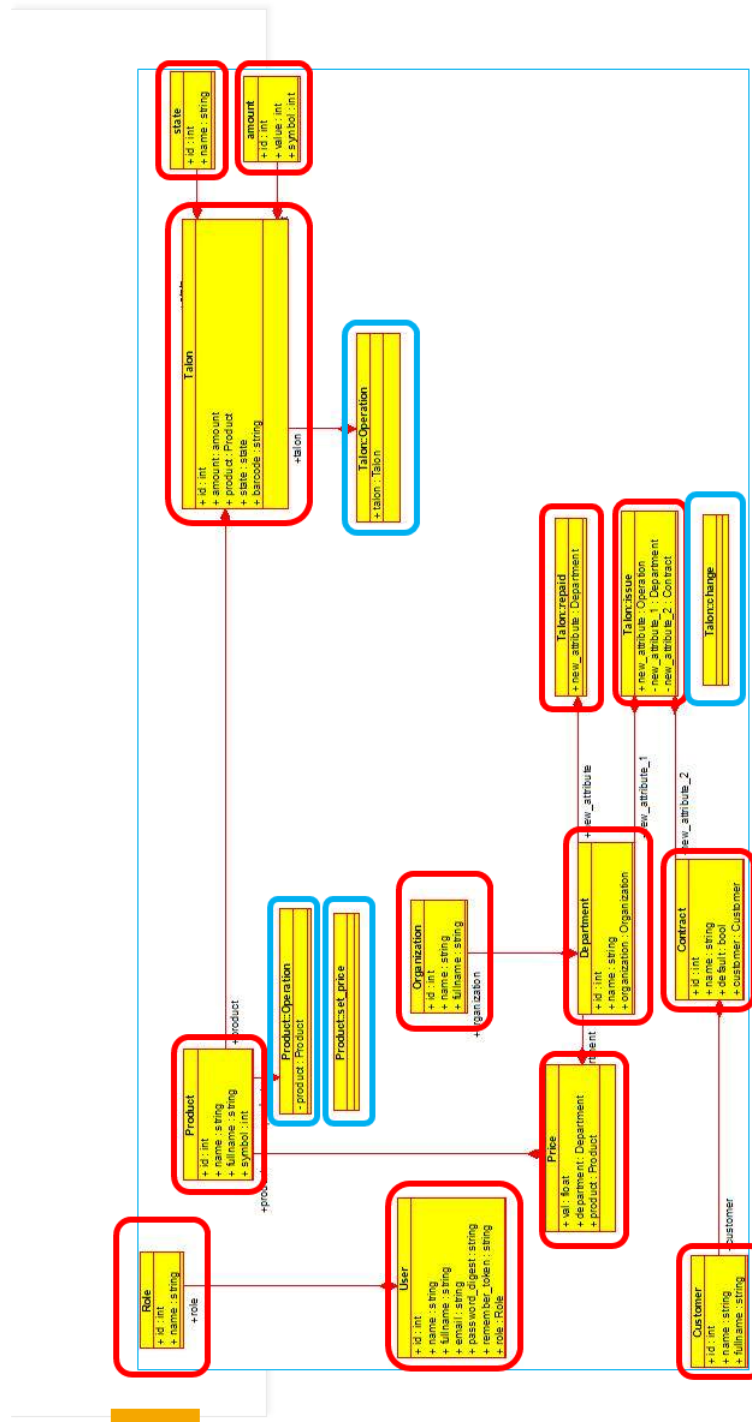


Figure B.23: Talon UML class Diagram

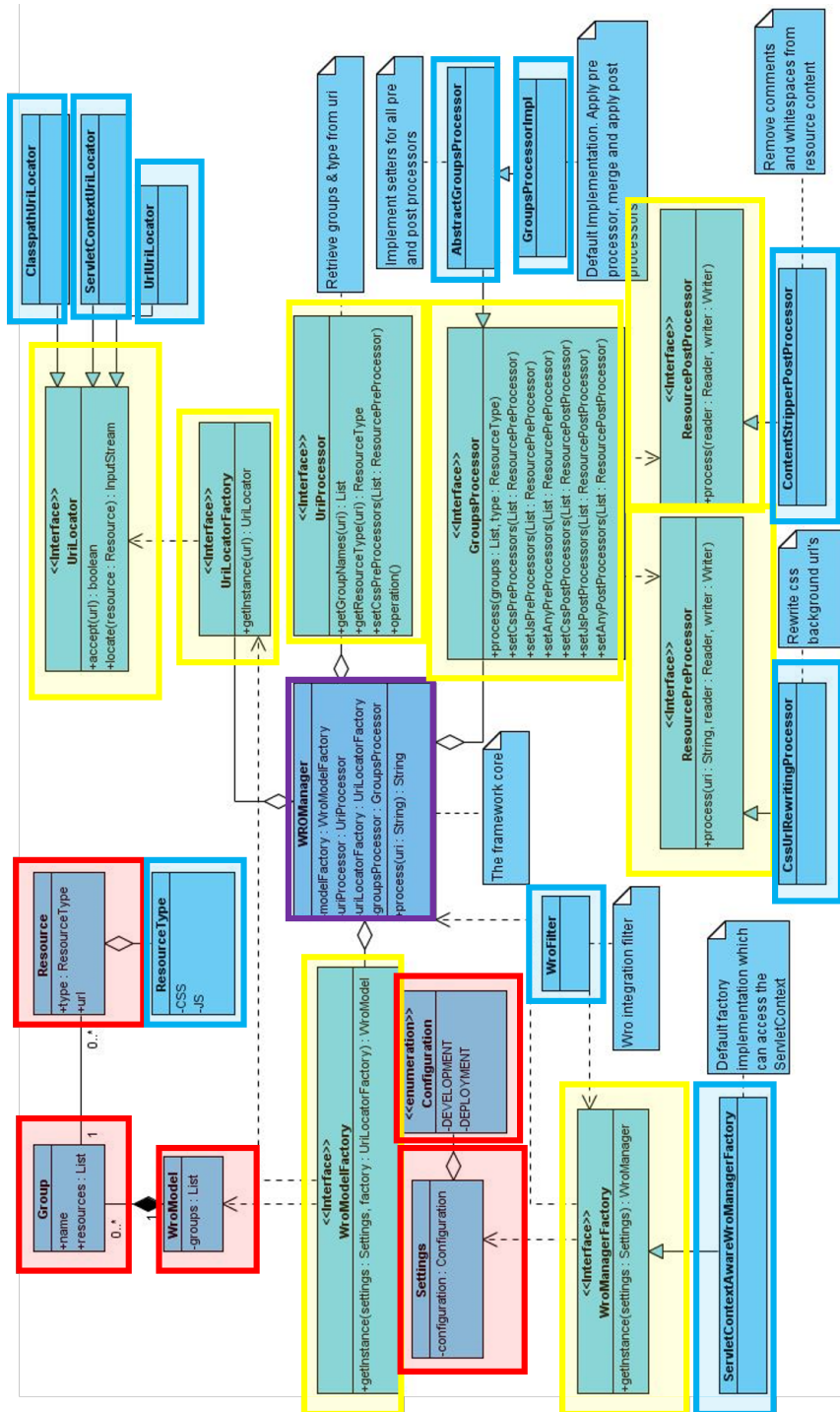


Figure B.24: Wro4j UML class Diagram

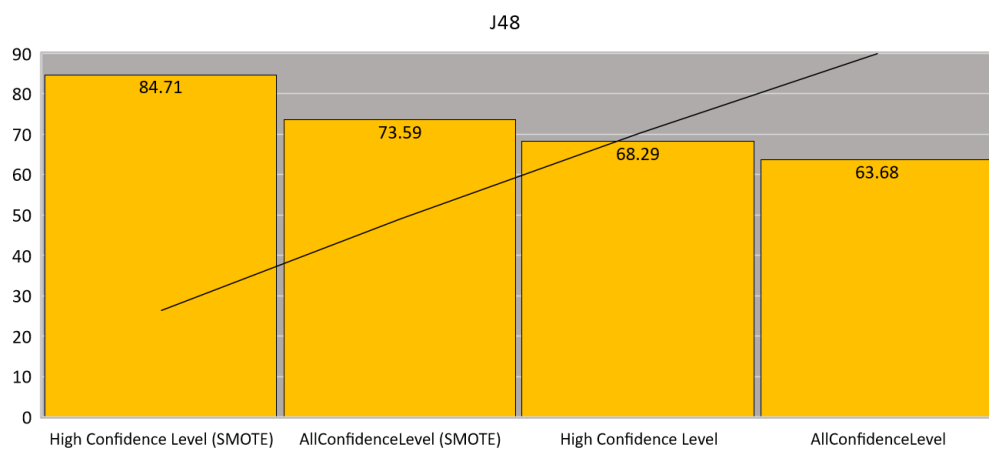




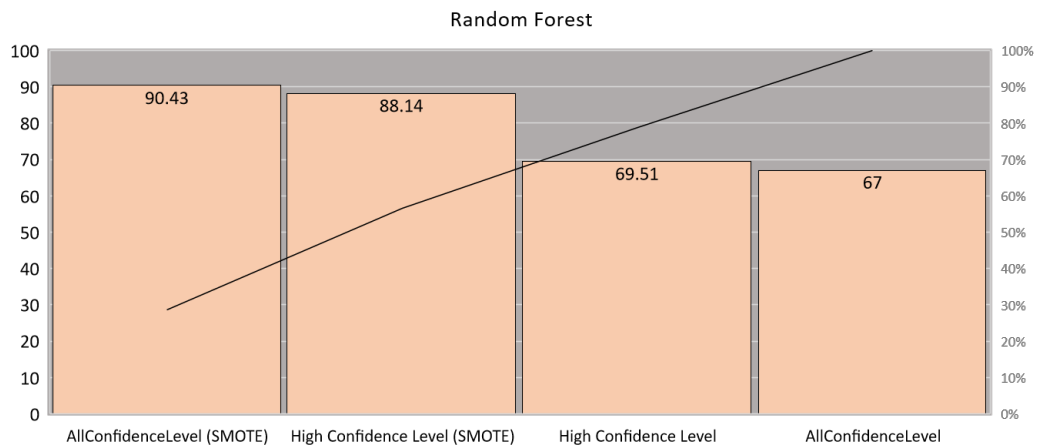
# C

## Appendix C

In this appendix, we have illustrated the classification accuracy of different ML algorithms for our datasets.



**Figure C.1:** *Accuracy of J48 classifier on different dataset*



**Figure C.2:** *Accuracy of Random Forest classifier on different dataset*

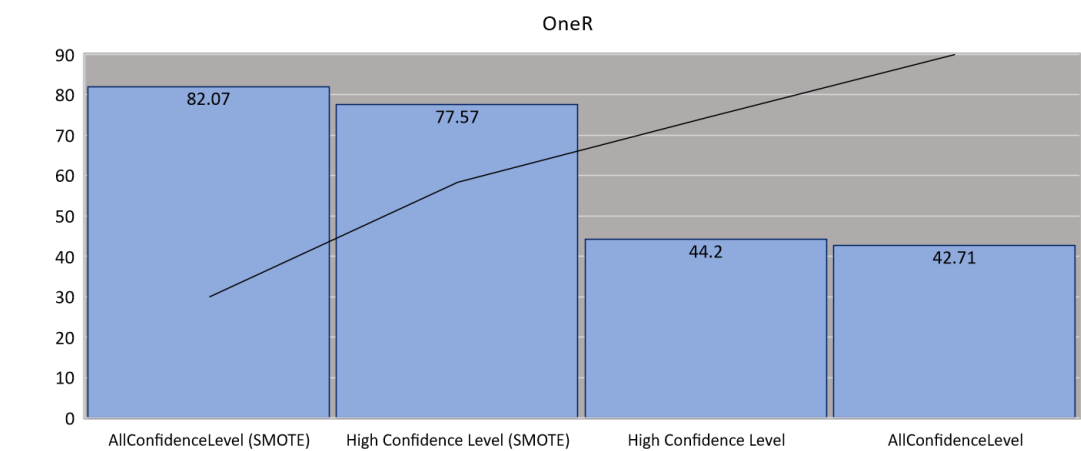


Figure C.3: Accuracy of OneR classifier on different dataset

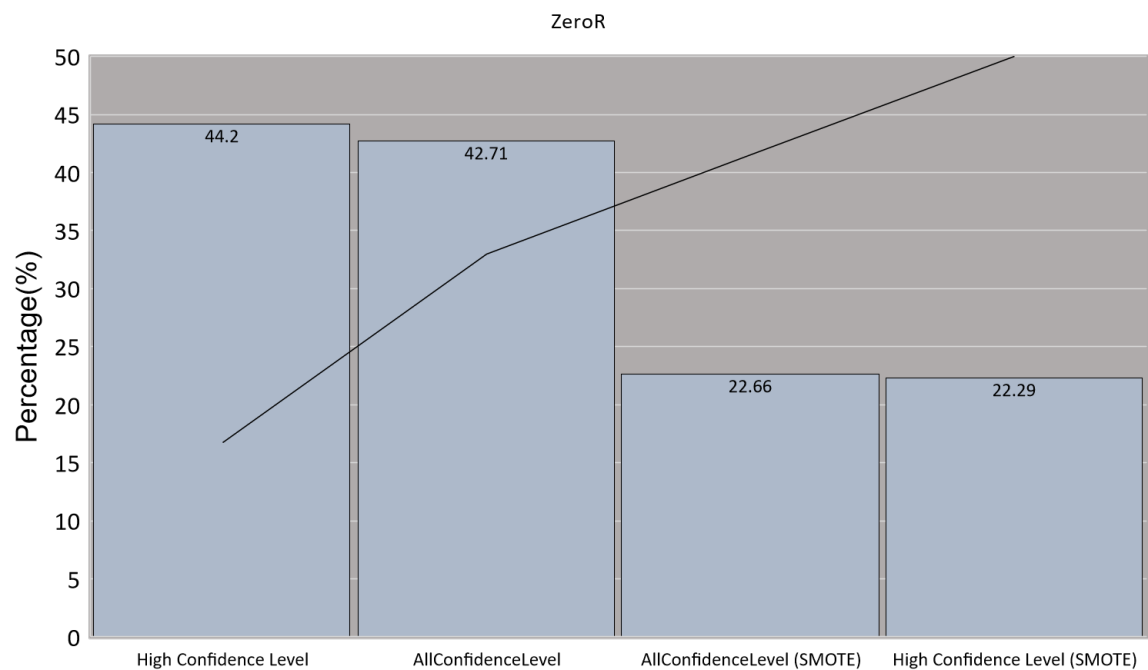


Figure C.4: Accuracy of ZeroR classifier on different dataset

# D

## Appendix D

In this appendix, we have illustrated some sequence diagrams from the K9 project.

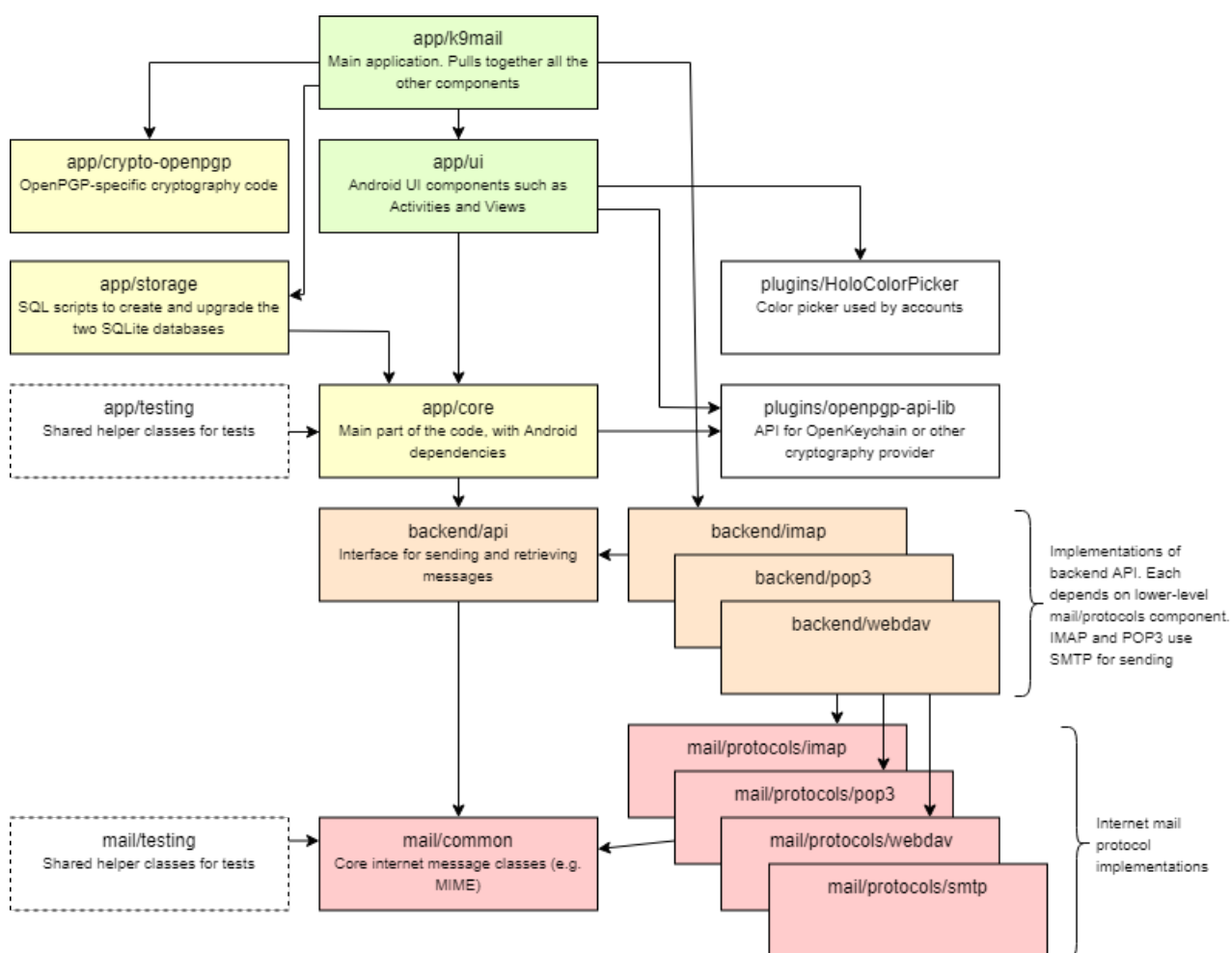


Figure D.1: K9 Project Sequence Diagram 1

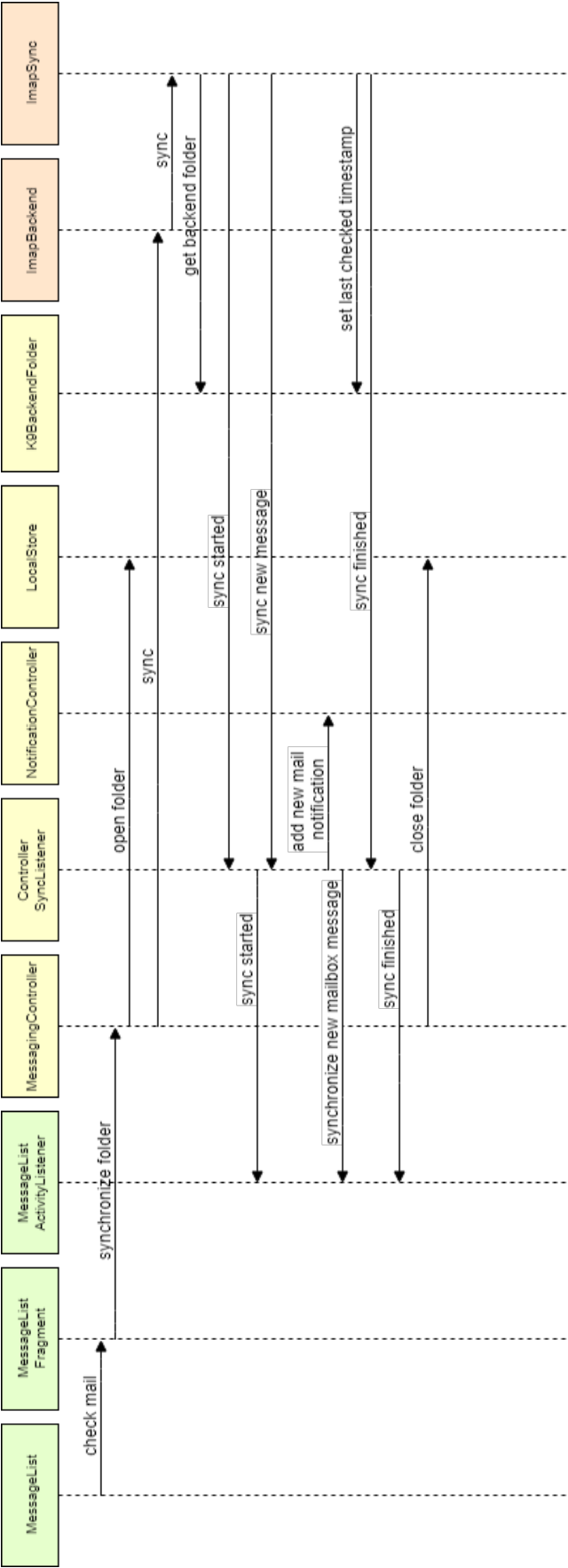


Figure D.2: K9 Project Sequence Diagram 2

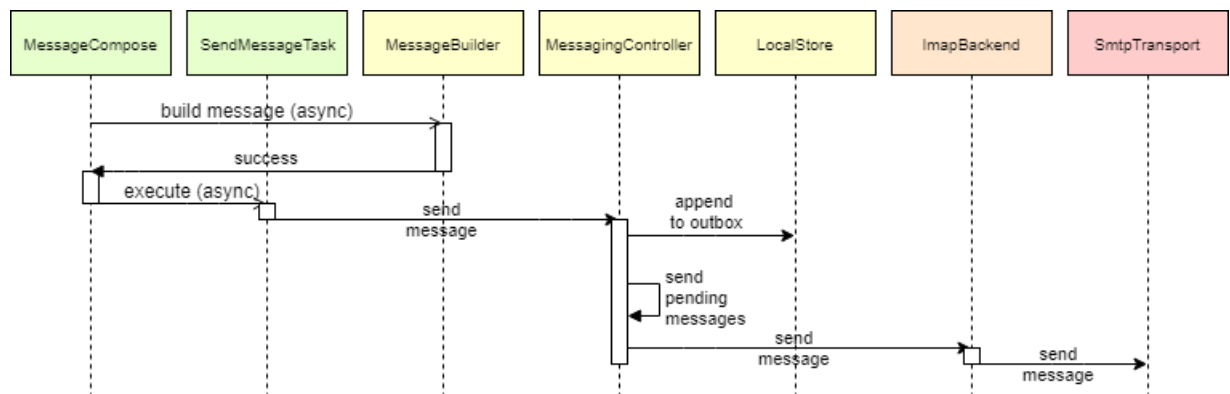


Figure D.3: *K9 Project Sequence Diagram 3*



# E

## Appendix E

In this appendix, we attach the charts that show the ranking of projects for each role stereotype. It is sorted in descending order.

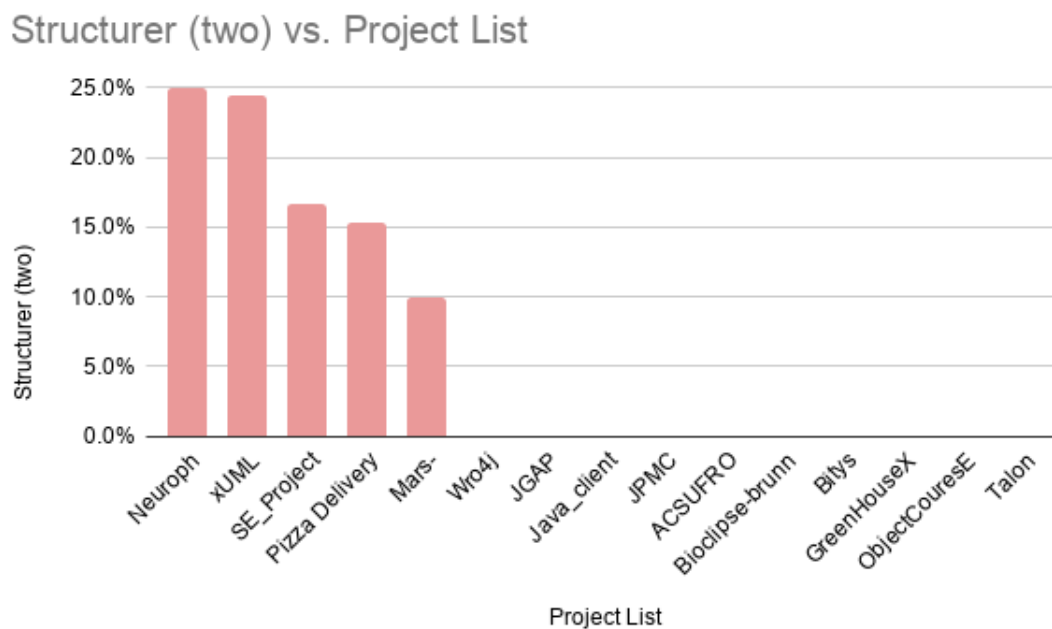


Figure E.1: *Structurer*

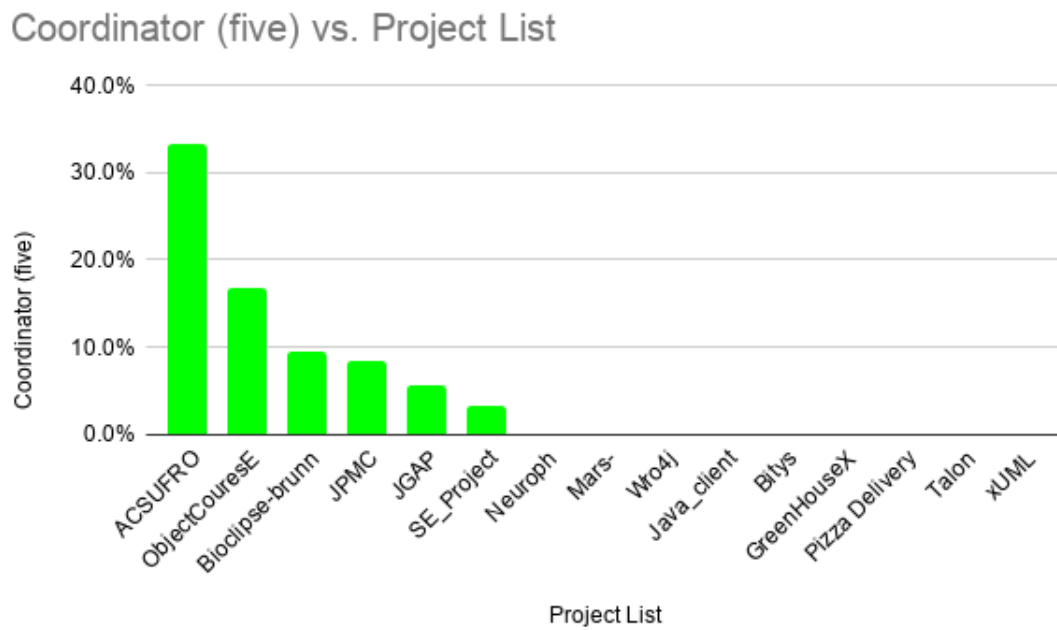


Figure E.2: *Coordinator*

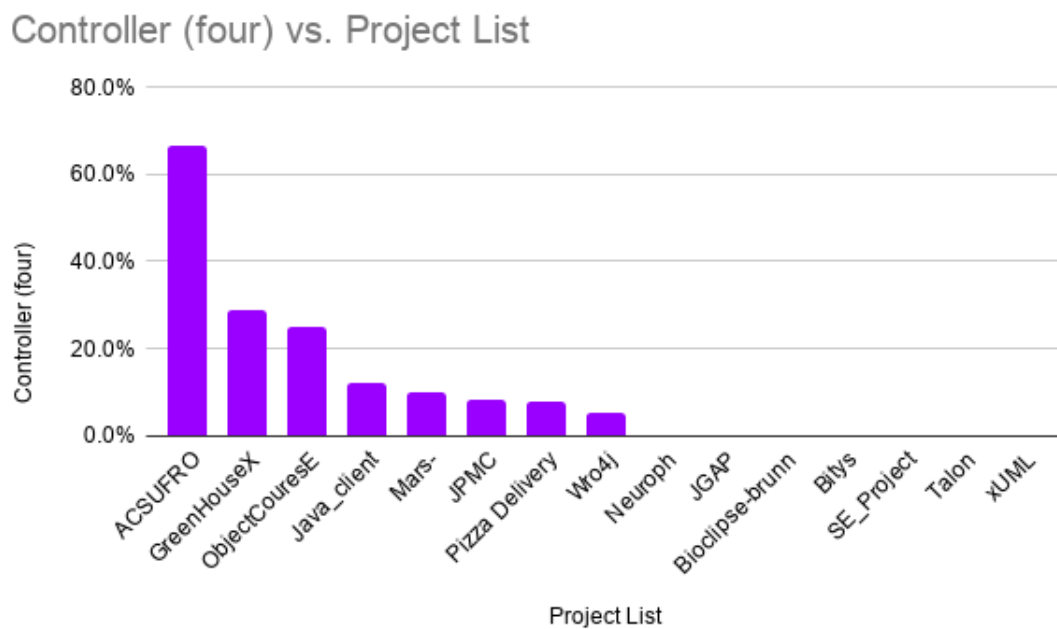
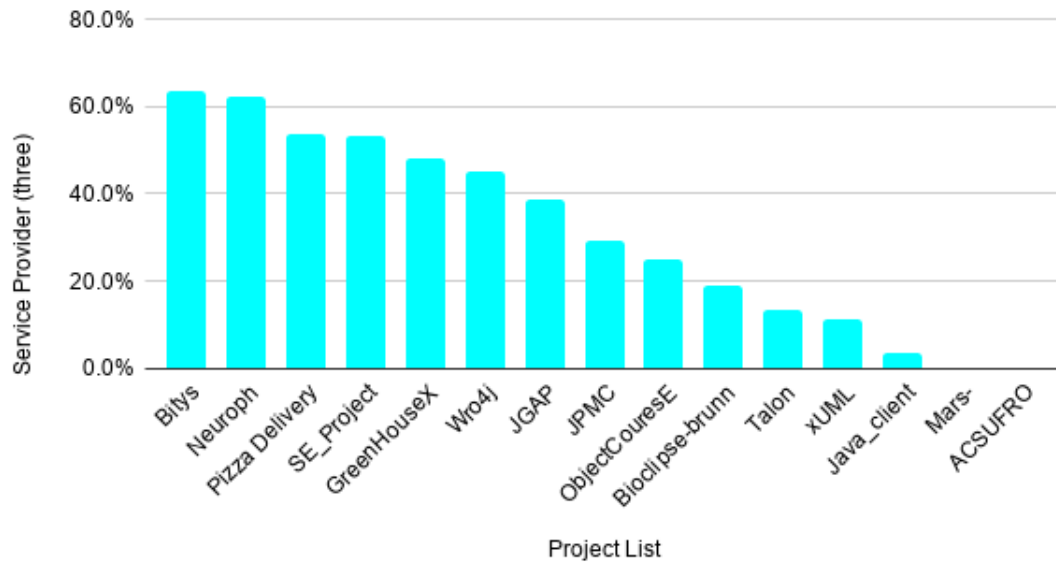


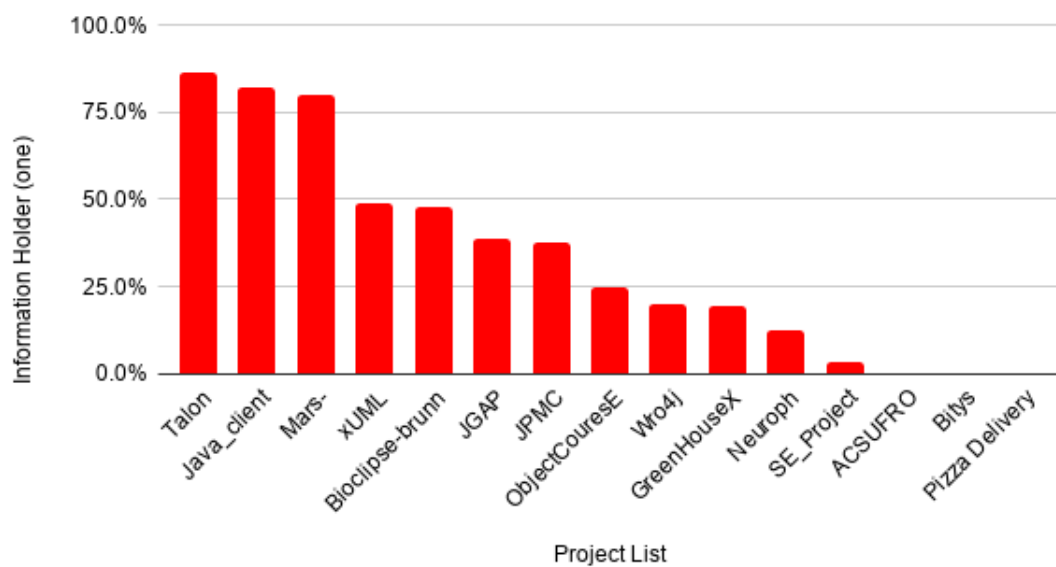
Figure E.3: *Controller*

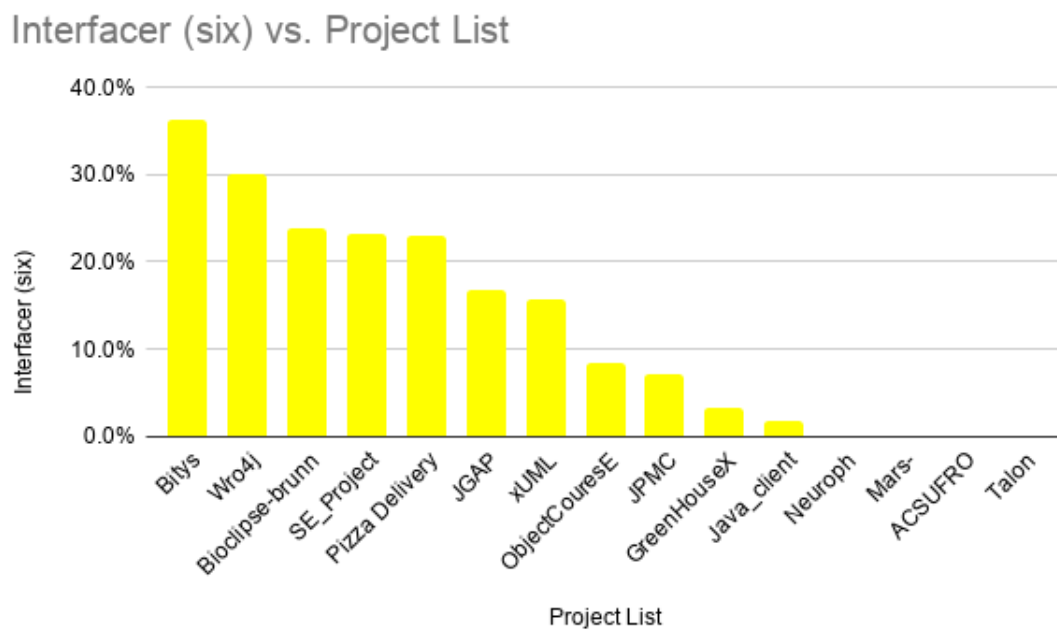


Service Provider (three) vs. Project List

Figure E.4: *ServiceProvider*

Information Holder (one) vs. Project List

Figure E.5: *InformationHolder*

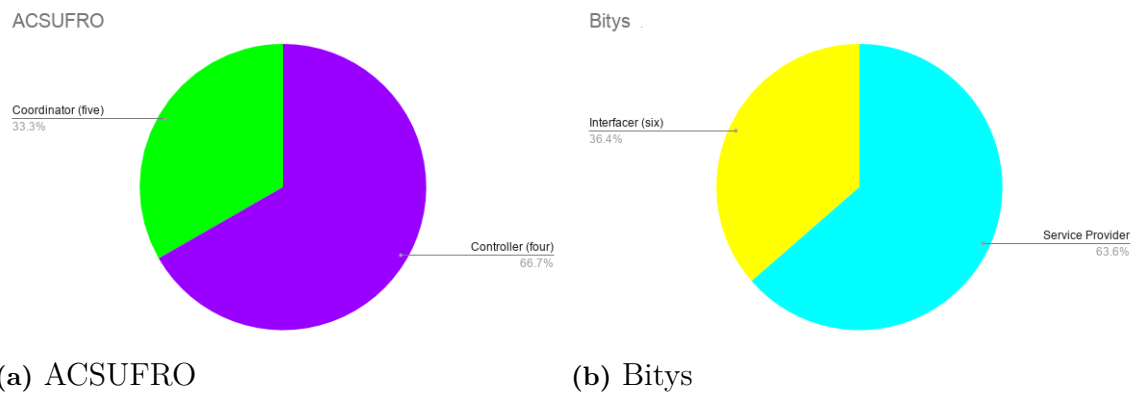


**Figure E.6:** *Interfacer*

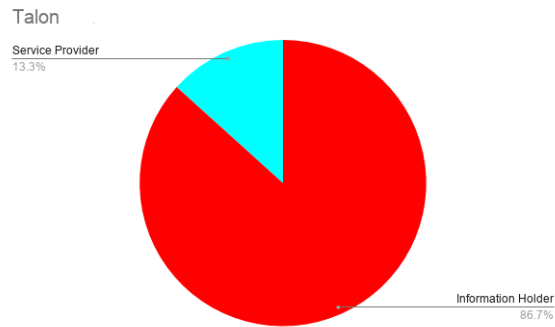
# F

## Appendix F

In this appendix, we attach the pie charts that show the percentage of role stereotypes for each project.

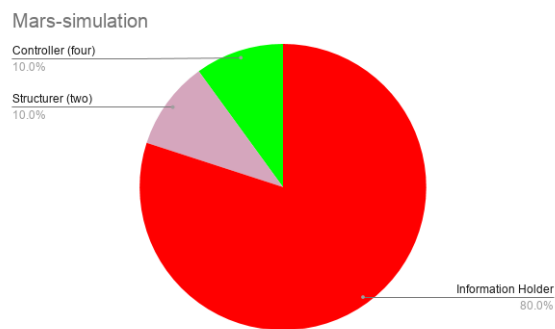


**Figure F.1:** Distribution of role stereotypes in ACSUFRO and Bitys

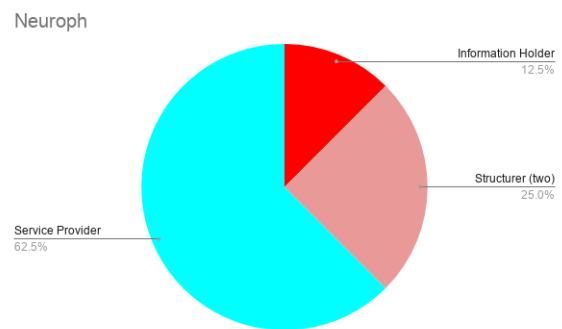


(a) Talon

**Figure F.2:** Distribution of role stereotypes in Talon

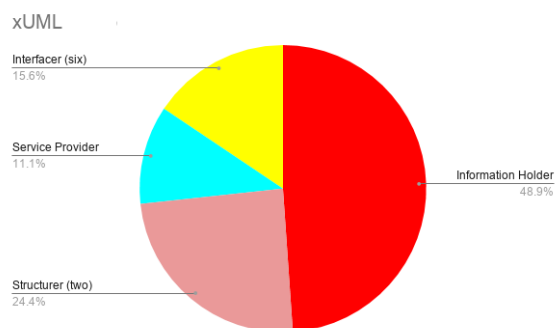


(a) Mars-sim

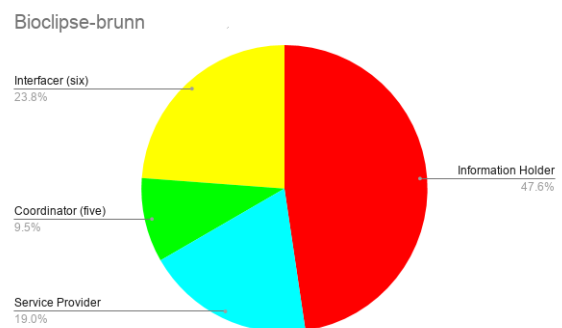


(b) Neuroph

**Figure F.3:** Distribution of role stereotypes in MarsSimulation and Neuroph

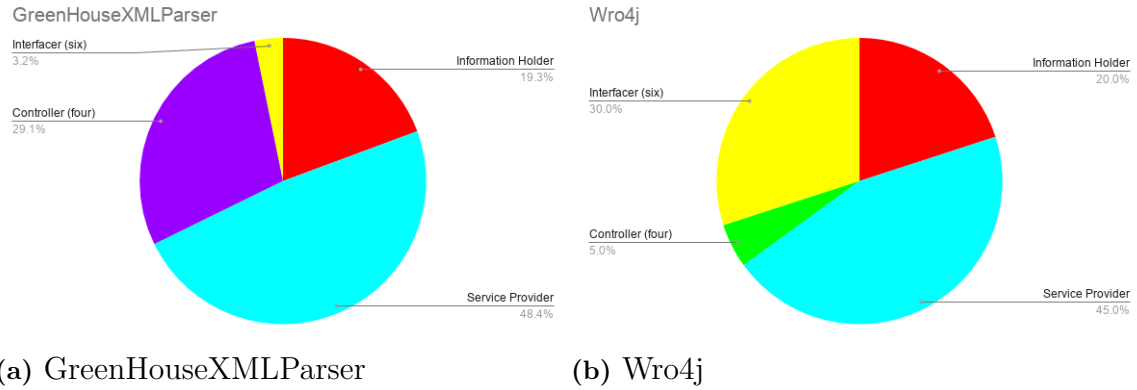


(a) xUML

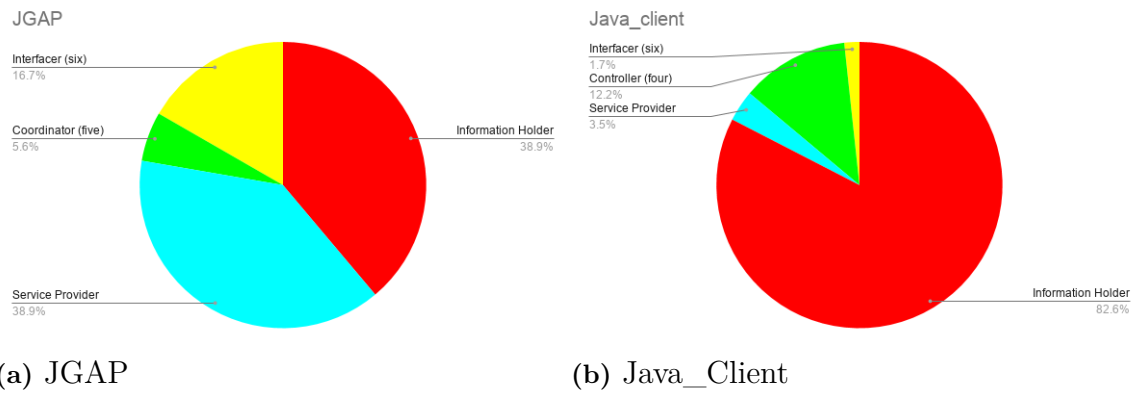


(b) Bioclipsebrunn

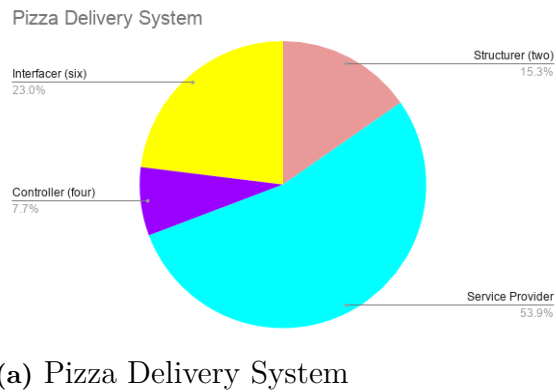
**Figure F.4:** Distribution of role stereotypes in xUML and BioClipsebrunn



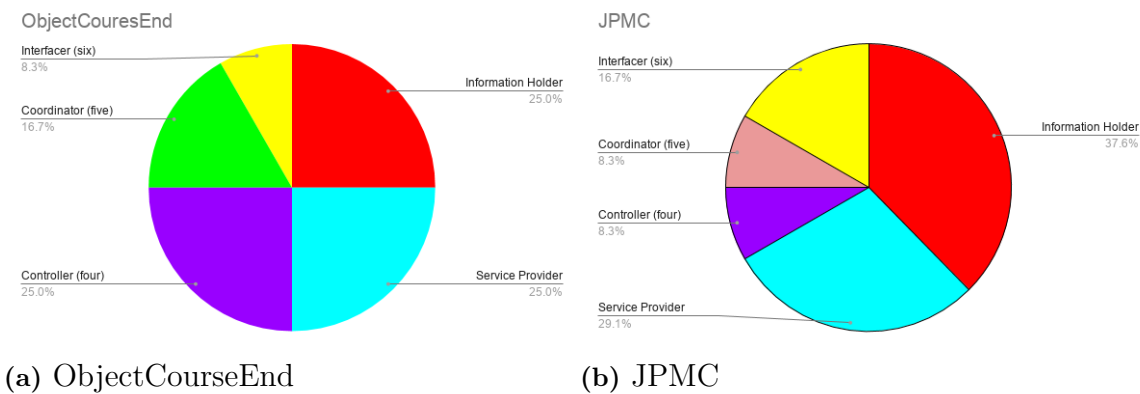
**Figure F.5:** Distribution of role stereotypes in XMLParser and Wroj4j



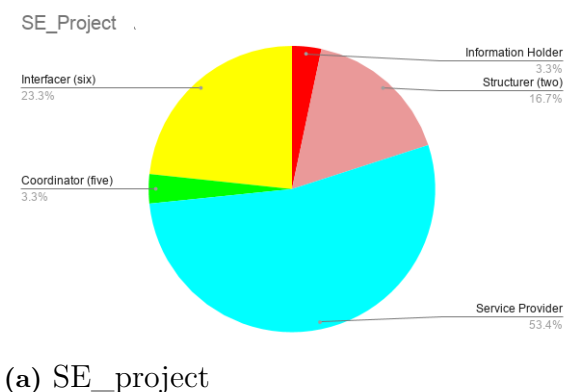
**Figure F.6:** Distribution of role stereotypes in JGAP and Java Client



**Figure F.7:** Distribution of role stereotypes in Pizza Delivery System



**Figure F.8:** Distribution of role stereotypes in ObjectCourseEnd and JPMC



**Figure F.9:** Distribution of role stereotypes in SE Project