Bad-smell Prediction from Software Design Model Using Machine Learning Techniques

Nakarin Maneerat
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University
Bangkok, Thailand
Email: n4nn@msn.com

Pornsiri Muenchaisri
Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University
Bangkok, Thailand
Email: Pornsiri.Mu@chula.ac.th

Abstract— **Bad-smell prediction significantly impacts on software quality. It is beneficial if bad-smell prediction can be performed as early as possible in the development life cycle. We present methodology for predicting bad-smells from software design model. We collect 7 data sets from the previous literatures which offer 27 design model metrics and 7 bad-smells. They are learnt and tested to predict bad-smells using seven machine learning algorithms. We use cross-validation for assessing the performance and for preventing over-fitting. Statistical significance tests are used to evaluate and compare the prediction performance. We conclude that our methodology have proximity to actual values.**

Keywords- Bad-smell; Software Design Model; Random Forest; Design Diagram Metrics; Prediction models; Machine Learners

I. INTRODUCTION

Generally, the software development life cycle activities consist of requirements analysis, design, implementation, testing, and maintenance. If we can predict software's quality as early as possible in the development life cycle, it will reduce the time of the development process. Bad-smell is one of the problems affecting the software's quality due to inefficient design or incomplete implementation. We could reduce time and man-hours for developing software if we have an approach for detection the bad-smells earlier.

Fowler and Beck [2] define 22 bad-smells and investigate the technique for removing the bad-smells. In the recent years, there are many automatic techniques used to detect bad-smells from source code. Munro [3], Radu Marinescu [4], Thisana Pienlert and Pornsiri Muenchaisri [5] introduce approaches for detecting, and locating some bad-smells by defining metric-based detection technique.

Khanti Yeesoon [6] proposes an approach for detecting and locating four bad-smells in Java programs by using Prolog rules. The results are compared with actual results.Sakorn Mekruksavanich [7] proposes a declarative-based approach for detecting design flaws of an object-oriented system. It can be detected at the meta-level in the declarative meta programming.

Figure 1 shows common approach part of the development life cycle that predict bad-smell from source code. Common approach is to create a program design and implement it into source code. Then we generate source code metrics for indicating whether source code contains the bad-smell or not. The development life cycle returns to design phase to redesign the program if there is a bad-smell in any classes.

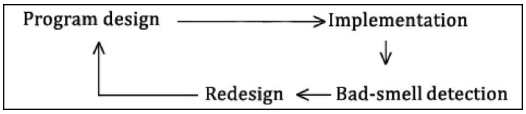

Figure 1. Typical life cycle with bad-smell detection from source code.

Literatures [3] [4] [5] [6] [7] propose approach to detect the bad-smells from source code. These approaches can detect bad-smell late in the development life cycle because source code have to be written before using their technique to detect the bad-smells. We use the concept of the paper which detects bad software properties from software design model. Figure 2 shows another approach to predict the bad-small earlier in development life cycle from software design model before code implementation.

Sallie Henry and Calvin Selig [1] predict source code complexity at the design stage to reduce the life cycle and increase the efficiency of the development process. They use linear regression for predicting the source code complexity. Then they compare the prediction results with actual result.

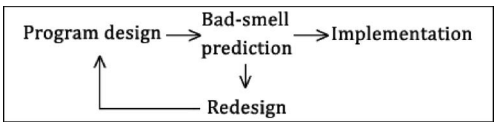

Figure 2. The life cycle with bad-smell prediction from software design model.

In this paper, we introduce an approach to predict bad-smells from software design model using machine learning techniques [9]. We use number of bad-smells and software design model metrics as data sets. The data sets are used as a learning set and testing set to predict bad-smells with 7 machine learning algorithms. We compare the performance of each machine learning algorithm by using several statistical significance tests such as prediction accuracy, hypothesis test, sensitivity and specificity, and predictive value of tests.

The remainder of this paper is organized as follows. Section 2 presents the related work. Section 3 describes description of bad-smells, design diagram metrics, machine learners and the statistical significance tests. Section 4 presents the experiments. Section 5 presents the experimental results and evaluation of the proposed approach. Section 6 discusses the experimental results. Section 7 presents the conclusion and future work.

II. RELATED WORK

In the last decade, many attentions have been paid to the quality of software. There are lots of techniques for error detection and software quality prediction. We categorize the techniques into two groups. There are software quality detection from source code and prediction from design model.

A. *Software quality detection from source code*

Munro [3] uses the bad-smell descriptions that are given by Fowler for identifying the characteristics of a bad smell. He defines a set of measurements and interpretation rules for bad-smells. He implement a prototype tool to evaluate the measurements and interpretation in case studies.

Radu Marinescu [4] presents approach for detecting design flaws using a metrics-based. He applies his technique on case studies and discusses the results. This approach shows its performance that can detect design flaws in the system.

Thisana Pienlert [5] presents an approach for detecting, locating and elimination six bad-smells from source code using software metrics. The metrics are used as indicators for determining the bad-smells. She evaluates her approach by comparing bad-smell metrics before and after applying the refactoring techniques. She confirms that her approach can be used as indicators for detecting and locating bad-smells.

Khanti Yeesoon [6] proposes an approach for detecting and locating four bad-smells by using Prolog rules. The approach has the following steps: designing Prolog rules from bad-smells; constructing Prolog facts from sample Java source code and searching bad-smells using Prolog query. To assess the approach, Prolog queries are used to detect bad-smells in Java programs. The results are compared with actual results.

Sakorn Mekruksavanich [7] proposes an approach for detecting design flaws of an object-oriented system. The design flaws can be detected at the meta-level in the declarative meta programming. He uses Prolog language for writing meta programming. He shows a design flaws detection example by applying his approach to detect high coupling flaws such as Message Chains, Inappropriate Intimacy and Middle Man.

B. *Software quality prediction from Design Model*

Sallie Henry [1] predict source code complexity at the design stage to reduce the life cycle and increase the efficiency of the development process. He uses Fan-in/Fan-out (software metric) to analyze the software quality. Then, he uses lines of code and software science to predict the source code's quality using linear regression analysis as the predictors.

III. METRICS & MACHINE LEARNERS DESCRIPTION

A. *Metrics description*

1) *Bad-smells*

Fowler and Beck define bad-smell as a feature that something has gone wrong somewhere in design software or code. It causes incomprehensible in software and difficulty in maintenance [2]. In this study we investigate 7 bad-smells. Bad-smells descriptions are listed in Table I.

TABLE I. BAD-SMELL DESCRIPTIONS

Bad-smell	Abbrev.	Definition
Lazy Class	LZC	Class is not doing enough or has less responsibility and has few functionality.
Feature Envy	FE	Class seems more interested in another class too much. That affects the design's flexibility.
Middle Man	MM	Class delegates between two or more classes.
Message Chains	MC	Class has long sequences of method calls or temporary variables to get routine data.
Long Method	LM	Method has many statements. So, that method is difficult to read, understand, and maintain.
Long Parameter Lists	LPL	Method is passed many parameters. The code more complex.
Switch Statement	SS	Method indicates the duplicate source code and reveals a lack of object orientation.

2) *Design Diagram Metric*

One of the well-known design diagram measurement is class diagram metrics that can evaluate quality of the software early. The class diagram is the key artifact in the OO programming paradigm that describes the structure of software by showing the software's classes, attributes, and the relationships among the classes.

In this study we use 27 software metrics, including Basic, Class Employment, Complexity, Diagrams, Inheritance, MOOD[10], Model size, other, and Relationship. Metric descriptions are listed in Table II.

The data sets are collected from previous literatures. It is extracted by using MagicDraw 9.5 SP1.1, which is the reverse engineering tool that offers UML models and model quantity from source code. It is not irregularity getting design metrics from code. Reverse engineering of quality measures has been used in many literatures [8], [13], [14]

TABLE II. METRIC DESCRIPTIONS USED IN THIS STUDY

Metric Type	Metric	Definition
Basic	NA	Number of attributes
Basic	NC	Number of Classes
Basic	NM	Number of Members
Basic	NO	Number of Operation
Basic	NP	Number of Parameters
Class Employment	C_PARAM	Number of times class is used as parameter type
Complexity	RFC	Response for a class
Complexity	WAC	Weighted attributes per class
Complexity	WMA	Weighted methods per class
Diagrams	D_APPEAR	Appearance in diagrams
Inheritance	DIT	Depth of inheritance tree
Inheritance	NOC	Number of children
Inheritance	NAI	Number of inherited attributes
Inheritance	NOI	Number of inherited operations
MOOD	AHF	Attribute hiding factor
MOOD	AIF	Attribute inheritance factor
MOOD	CF	Coupling factor
MOOD	MHF	Method hiding factor
MOOD	MIF	Method inheritance factor
MOOD	PF	Polymorphism factor
Model size	ACT	Number of actors
Model size	COMP	Number of components
Model size	NS	Number of namespaces
Other	CBC	Coupling between objects
Relationship	ABSTR_R	Number of abstractions
Relationship	ASSOC_R	Number of associations
Relationship	DEPEND_R	Number of dependencies

B. Machine learners

Machine learners are computational methods and construction of programs that automatically learn to recognize complex patterns and make intelligent decisions based on data.

Weka [11] is the popular suite of machine learning algorithms for data mining tasks. It contains 37 algorithms for classification problems (Weka 3.6).

In this study, we compare the performance of Random Forest with other machine learning algorithms shown in Table III such as Naive Bayes, Logistic, IB1, IBk, VFI and J48 from Weka machine learning software suite on the same data sets obtained from previous literatures. The 7 machine learning algorithms are used with their default parameters.

TABLE III. MACHINE LEARNERS USED IN THIS STUDY

	Learner	Abbrev.
1	Random Forest	RF
2	Naive Bayes	NB
3	Logistic regression	LGI
4	IB1	IB1
5	IBk	IBk
6	VFI	VFI
7	J48	J48

1) Random Forest: Random forest(RF)[9] is a classifier composed by a collection of many decision trees that each tree depends on the values of a random vector. The advantages of random forest are an ability to handle a very large number of input variables and to estimate the importance of variables in determining classification.

2) Naive Bayes: Naive Bayes(NB) is a simple probabilistic classifier technique using Bayesian theorem. Naive Bayes classifier can build model quickly and it also requires a small training data set to estimate the necessary parameters for classification.

3) Logistic regression: Logistic regression (LGI) is used to predict a categorical variable from a set of predictor variables. It describes the relationship among the categorical variable and predictor variables which use mathematical logistic regression function. Logistic regression is one of the best probabilistic classifiers.

4) IB1: IB1 (IB1) stands for Instance-Based1 learner that uses a simple distance measure to find the training instance nearest for the class of the test instances. If multiple instances have the same (smallest) distance to the test instance, the first one found is used.

5) IBk: IBk (IBk) is simple instance-based learner that uses a simple distance measure to find the nearest k training instances for the class of the test instances.

6) VFI: VFI (VFI) is classifier implementing the voting feature interval classifier. Upper and lower boundaries are constructed around each class. Class counts are recorded for each interval on each attribute. Classification is made by voting.

7) J48: J48 (J48) is a non-parametric algorithm used to generate a decision tree. The decision trees generated by J48 can be used for classification.

C. Cross-validation

We use 10-fold cross-validation approach to assessing the performance of a predictive model and preventing over-fitting of the data sets. The 10-fold cross-validation works as follows:

1) Separate data in to 10 segments (or folds)

2) Select the first segment for testing, while the remaining segments are used for training.

3) Perform classification and obtain performance metrics.

4) Select the next segment as testing and use the rest as training data.

5) Repeat classification until each segment has been used as the testing set. All segments have to cross-over in 10 rounds so that each segment has a chance to be validated.

6) Calculate an average performance from the individual experiments.

Kohavi [12] estimates accuracy of cross-validation by comparing several approaches. Those cross-validations are regular cross-validation, leave-one-out cross-validation, K-fold cross-validation and bootstrap. 10-fold cross-validation is the best model that has less bias in accuracy estimation.

D. Statistical significance test

We use the prediction accuracy estimators for comparing the performance of prediction by statistical significance tests.

1) Bad-smell prediction accuracy

Bad-smell prediction accuracy is given in the Weka output. It gives a measure for the overall accuracy of the classifier

$$\text{Accuracy}=\frac{\text{number of correctly classified instances}}{\text{number of instances}}$$

2) Specificity and sensitivity

- Specificity is the proportion of true positives that are correctly predicted (The accuracy of the prediction of the classes have bad-smell).

$$\text{Specificity}=\frac{\text{True positive}}{\text{True positive}+\text{False negative}}$$

- Sensitivity is the proportion of true negatives that are correctly predicted (The accuracy of the prediction of the classes have no bad-smell).

$$\text{Sensitivity}=\frac{\text{True negative}}{\text{True negative}+\text{False positive}}$$

3) Predictive value of tests

- Positive predictive value (PPV) is the prediction accuracy with positive test results that shows the accuracy of the classes' prediction as there is bad-smell in classes.

$$\text{PPV}=\frac{\text{True positive}}{\text{True positive}+\text{False positive}}$$

- Negative predictive value (NPV) is the prediction accuracy with negative test results that shows the accuracy of the classes' prediction as there is no bad-smell in classes.

$$\text{NPV}=\frac{\text{True negative}}{\text{True negative}+\text{False negative}}$$

IV. BAD-SMELL PREDICTION

We now present a description of the bad-smell prediction and experimental process as following steps.

1) In our case studies, we collect the data sets from software of previous literatures that have bad-smells. The prediction variables are the number of actual bad-smells in each class, including LZC, FE, MM, MC, LM, LPL and SS.

2) We use class diagram metrics that we define in previous section as the input variables. Each data set from the previous literatures is extracted into the number of class diagram metrics by using MagicDraw 9.5 SP1.1.

3) We use 7 machine learning algorithms from Weka for prediction of bad-smells on each data set. There are Naive Bayes, Logistic, IB1, IBk, VFI, J48 and Random forest.

- Each data set that contains each bad-smell is learned and tested by 7 machine learning algorithms with their default parameters.
- We apply 10-fold cross-validation for estimating generalization error. It is the one of Wake's features that divides data set into 10 segments. It uses 9 segments to train and uses another to validate. Each segment has to be validated by cross-over segments in 10 rounds

In our methodology, 7 data sets that contained each bad-smell are learned by 7 machine learning algorithms. There are 49 (7X7) result sets from the experiments.

4) After that, we compare the performance of bad-smells prediction from software design model with the actual value. We also compare the performance of each machine learning algorithm that are described and evaluated in the next section.

V. EXPERIMENTAL RESULTS

In this section, we present the results of our experiments that show the performance of bad-smells prediction with 7 machine learner's algorithms such as Random Forests, Naive Bayes, Logistic, IB1, IBk, VFI and J48 and report the bad-smell prediction accuracy, the specificity and sensitivity, and the predictive value of tests.

A. Bad-smell prediction accuracy

Table IV shows prediction accuracy of bad-smell prediction from software design model by machine learners. We use 10-fold cross-validation to evaluate the accuracy. The 10-fold cross-validation is run for at least 10 times in each experiment with different machine learning algorithms. We can observe the following trends:

- Logistic regression, IB1, IBk and Random Forest have a better performance than others. They are able to achieve over 90% bad-smell predation average rate.
- Every machine learners is good for prediction Switch Statement bad-smell.

B. Specificity and sensitivity

Now, we use Specificity and sensitivity to measure the performance of bad-smell prediction on 49 result sets.

Table V shows specificity of bad-smells prediction. The performance of IB1, IBk and VFI is better than that of the others. They have the average of nearly 90% specificity.

Table VI shows sensitivity of bad-smells prediction. The performance of Logistic regression, IB1, IBk, J48 and Random Forest is better than that of the Naive Bayes and VFI. They have the average of nearly 90% specificity.

C. Predictive value of tests

Table VII shows PPV of bad-smells prediction from software design model. The Logistic regression, IB1, IBk and Random Forest are better than the performance of the others. They have average of more than 80% PPV.

Table VIII shows NPV of bad-smells prediction from software design model. Naive Bayes, Logistic regression, IB1, IBk, Random Forest are better than the performance of the VFI and J48. They have average of more than 90% PPV.

TABLE IV. BAD-SMELL PREDICTION ACCURACY

	FE (%)	LM (%)	LPL (%)	LZC (%)	MC (%)	MM (%)	SS (%)	Average (%)
Naive Bayes	72.77	85.34	73.30	73.30	74.07	59.26	92.15	78.38
Logistic regression	95.29	93.19	95.81	95.81	92.59	81.48	96.86	93.81
IB1	96.34	95.81	95.81	95.81	92.59	85.19	96.86	94.74
IBk	95.29	96.34	96.86	96.86	92.59	85.19	97.38	95.00
VFI	72.25	82.72	70.68	70.68	74.07	48.15	84.29	72.65
J48	94.24	93.72	97.38	97.38	62.96	62.96	98.43	88.32
Random Forest	96.86	95.29	96.86	96.86	92.59	81.48	96.86	94.53

TABLE V. BAD-SMELL PREDICTION SPECIFICITY

	FE (%)	LM (%)	LPL (%)	LZC (%)	MC (%)	MM (%)	SS (%)	Average (%)
Naive Bayes	88.89	100.00	100.00	95.00	66.67	66.67	94.74	87.42
Logistic regression	88.89	77.27	50.00	75.00	100.00	93.33	89.47	82.00
IB1	88.89	86.36	90.91	80.00	100.00	93.33	89.47	89.85
IBk	88.89	90.91	90.91	75.00	100.00	93.33	89.47	89.79
VFI	100.00	90.91	95.45	100.00	83.33	86.67	94.74	93.01
J48	0.00	77.27	86.36	75.00	50.00	26.67	89.47	57.83
Random Forest	88.89	90.91	95.45	75.00	50.00	86.67	89.47	82.34

TABLE VI. BAD-SMELL PREDICTION SENSITIVITY

	FE (%)	LM (%)	LPL (%)	LZC (%)	MC (%)	MM (%)	SS (%)	Average (%)
Naive Bayes	73.63	85.21	58.58	71.93	80.00	58.33	91.86	74.22
Logistic regression	97.25	96.45	99.41	98.25	86.67	83.33	97.67	94.15
IB1	98.35	97.63	98.22	97.66	86.67	83.33	97.67	94.22
IBk	98.35	97.63	98.22	99.42	86.67	83.33	98.26	94.55
VFI	73.08	82.25	54.44	67.25	66.67	8.33	83.14	62.16
J48	98.90	97.04	98.22	100.00	73.33	100.00	99.42	95.27
Random Forest	99.45	96.45	99.41	99.42	86.67	66.67	97.67	92.25

TABLE VII. POSITIVE PREDICTIVE VALUE OF BAD-SMELL PREDICTION

	FE (%)	LM (%)	LPL (%)	LZC (%)	MC (%)	MM (%)	SS (%)	Average (%)
Naive Bayes	14.29	46.81	23.91	28.36	72.73	66.67	56.25	44.14
Logistic regression	61.54	73.91	91.67	83.33	85.71	87.50	80.95	80.66
IB1	72.73	82.61	86.96	80.00	85.71	87.50	80.95	82.35
IBk	72.73	83.33	86.96	93.75	85.71	87.50	85.00	85.00
VFI	15.52	40.00	21.43	26.32	66.67	54.17	38.30	37.48
J48	0.00	77.27	86.36	100.00	60.00	100.00	94.44	74.01
Random Forest	88.89	76.92	95.45	93.75	75.00	76.47	80.95	83.92

TABLE VIII. NEGATIVE PREDICTIVE VALUE OF BAD-SMELL PREDICTION

	FE (%)	LM (%)	LPL (%)	LZC (%)	MC (%)	MM (%)	SS (%)	Average (%)
Naive Bayes	99.26	100.00	100.00	99.19	75.00	58.33	99.37	90.17
Logistic regression	99.44	97.02	93.85	97.11	100.00	90.91	98.82	96.74
IB1	99.44	98.21	98.81	97.66	100.00	90.91	98.82	97.69
IBk	99.44	98.80	98.81	97.14	100.00	90.91	98.83	97.71
VFI	100.00	98.58	98.92	100.00	83.33	33.33	99.31	87.64
J48	95.24	97.04	98.22	97.16	64.71	52.17	98.84	86.20
Random Forest	99.45	98.79	99.41	97.14	68.42	80.00	98.82	91.72

TABLE IX. SUITABLE MECHINE LERNER FOR BAD-SMELLS BY MEASUREMENT METHOD

	Prediction accuracy	Specificity	Sensitivity	PPV	NPV
Feature Envy	RF	VFI	RF	RF	VFI
Long Method	IBk	NB	IB1 and IBk	IBk	NB
Long Parameter Lists	J48	NB	LGI and RF	RF	NB
Lazy Class	J48	VFI	J48	J48	VFI
Message Chains	LGI,IB1, IBk and RF	LGI, IB1 and IBk	LGI, IB1, IBk and RF	LGI, IB1 and IBk	LGI, IB1 and IBk
Middle Man	IB1 and IBk	LGI, IB1 and IBk	J48	J48	LGI, IB1 and IBk
Switch Statement	IBk	NB and VFI	J48	J48	NB

VI. DISCUSSION

The goal of this paper is to identify bad-smells as early as possible in the development life cycle. We discuss the performance of the prediction approaches by the measurement method and appropriate to use. Table V-VIII show prediction sensitivity, prediction specificity and predictive value of tests that there is no best machine learning algorithm that can accurately predict all bad-smells because some machine learning algorithm achieve 100% for some bad-smell. Table IX shows the suitable machine learner for detecting the bad-smells by measurement method.

A. Bad-smell prediction accuracy

Bad-smell prediction accuracy is the good measurement method for evaluate the results for predicting the bad-smell correctly. Naive Bayes, VFI and J48 are not suitable for software quality prediction because of their low bad-smell prediction accuracy. They achieve less than 90% bad-smell prediction average rate shown in Table IV. Although, most of them are able to achieve more than 90% for Long Method prediction but they have low prediction rate for Middle Man. Compared with these methods, Random Forests can achieve higher overall performance. Random Forests is as well as Logistic regression, IB1 and IBk because their predicted values have proximity to actual values for every data set.

B. Specificity and sensitivity

In case any project contain plenty of bad-smell classes, we should select the machine learners that offer high prediction specificity. Table V shows specificity of the approaches' values of each bad-smell. We can conclude that:

- Naive Bayes is suitable to predict the Long Method, Long Parameter and Switch Statement bad-smells.
- Logistic regression, IB1 and IBk are suitable to predict the Message Chains and Middle Man bad-smells.
- VFI is suitable to predict the Feature Envy and Lazy Class and Switch Statement bad-smells.

On the other hand, in case any projects contain scarcity of bad-smell classes, we select the machine learners that have high prediction sensitivity. Table VI shows sensitivity of approaches' values of each bad-smell. We can conclude that:

- Random Forest is suitable to predict the Feature Envy, Long Parameter Lists and Message Chains bad-smells.

- Naive Bayes is suitable to predict the Lazy Class, Message Chains and Switch Statement bad-smells.
- IB1 and IBk are suitable to predict the Long Method and Message Chains bad-smells.
- Logistic regression is suitable to predict the Long Parameter Lists and Message Chains bad-smells.

C. *Predictive value of tests*

In case any projects need the accuracy of the prediction bad-smell in class, PPV is suitable statistical method to analyze the good machine learners. Table VII shows PPV of approaches of each bad-smell. We can conclude that:

- Random Forest is suitable to predict the Feature Envy and Long Parameter Lists bad-smells.
- J48 is suitable to predict the Lazy Class, Middle Man and Switch Statement bad-smells.
- IBk is suitable to predict the Long Method and Message Chains bad-smells.
- Logistic regression and IB1 are suitable to predict the Message Chains bad-smells.

On the other hand, in case any projects need to prevent the good class becomes bad-smell class by the error of the prediction, NPV is suitable statistical method to analyze the good machine learners. Table VIII shows NPV of approaches of each bad-smell. We can conclude that:

- Naive Bayes is suitable to predict the Long Method, Long Parameter Lists and Switch Statement bad-smells.
- VFI is suitable to predict the Feature Envy and Lazy Class bad-smells.
- Logistic regression, IB1 and IBk are suitable to predict the Message Chains and Middle Man bad-smells.

VII. Conclusions and Future Work

The goal of this paper is to present a methodology for predicting bad-smells from software design model. We apply machine learning algorithms on 7 bad-smell data sets. Each data set has each bad-smell and its design metrics to predict each bad-smell. We evaluate bad-smell prediction accuracy with many statistical tests. Then, we compare the performance of machine learning algorithms. Our experiments show that bad-smell prediction from software design model could predict bad-smell earlier and they have proximity to actual values.

We evaluate machine learning algorithms from their bad-smell prediction accuracy. The results show that some algorithms are not suitable for bad-smell prediction because of their low bad-smell prediction rate. They are Naive Bayes, VFI and J48 are able to achieve less than 90% bad-smell predation average rate. Logistic regression, IB1, IBk and Random Forest can achieve higher overall prediction rate since they are able to achieve over 90% bad-smell predation average rate.

Sensitivity and specificity, and predictive value of tests are used to evaluate accuracy. We can conclude that there is no best machine learning algorithm that can accurately predict all selected bad-smells. We should consider the machine learning algorithm based on type of bad-smell.

There are some issues that could be made to improve the research. Our future work will focus on the following topics:

1) We can change or increase the size of the class diagram metrics to measure. We also can further research using other machine learning algorithms to show what is the best machine learning algorithms and the suitable class diagram metrics for bad-smell prediction.

2) There are other bad-smells remain to be researched. We may also research on how is the relationship among each bad-smell affects the prediction accuracy.

3) We use 7 learning algorithms with their default parameters. We can adjust any parameters to make the best experiment results.

References

[1] S. Henry, and C. Selig, "Predicting Source-Code Complexity at the Design Stage", IEEE Software, March 1990.

[2] M. Fowler. Refactoring: Improving the Design of Existing Code. United States: Addison-Wesley, 1999.

[3] M. Munro, "Product Metrics for Automatic Identification of "Bad Smell" Design Problems in Java Source-Code".11th IEEE International Software Metrics Symposium (METRICS 2005), 2005.

[4] R. Marinescu, "Detecting Design Flaws via Metrics in Object-Oriented Systems", Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39), August 2001.

[5] T. Pienlert and P. Muenchaisri, "Bad-smell Detection using Object-Oriented Software Metrics". International Society of Computers and Their Applications (ISCA) International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications (CSITeA'04), December 2004.

[6] K. Yeesoon, Design and implementation of a tool for detecting bad smells in Java program , Chulalongkorn University, 2008.

[7] S. Mekruksavanich and P. Muenchaisri, "Using Declarative Meta Programming for Design Flaws Detection in Object-Oriented Software", The 2009 International Conference on Computer Design and Applications (ICCDA2009), May 2009.

[8] Y. Jiang, B. Cukic, T. Menzies, and N. Bartlow, "Comparing Design and Code Metrics for Software Quality Prediction". Proceedings of the PROMISE 2008 Workshop (ICSE) ,2008.

[9] L. Breiman, "Random Forests", in Machine Learning, vol 45, pp.5-32, October 2001.

[10] B. Abreu F., L. Ochoa , and M. Goulao, "The MOOD metrics set", INESC/ISEG Internal Report, 1998.

[11] I.H. Witten, and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, October 1999, http://www.cs.waikato.ac.nz/ml/weka/ .

[12] R. Kohavi, A "study of cross-validation and bootstrap for accuracy estimation and model selection". in Proceedings of International Joint Conference on AI. 1995.

[13] G. Antoniol, G. Canfora, G. Casazza, A.D. Lucia, and E. Merlo, "Recovering tracebility links between code and documentation". IEEE Transactions on Software Engineering, 2002.

[14] G. Antoniol, G. Casazza, M. Penta, and R. Fiutem, "Object-oriented design patterns recovery". Journal of Systems and Software, 2001.