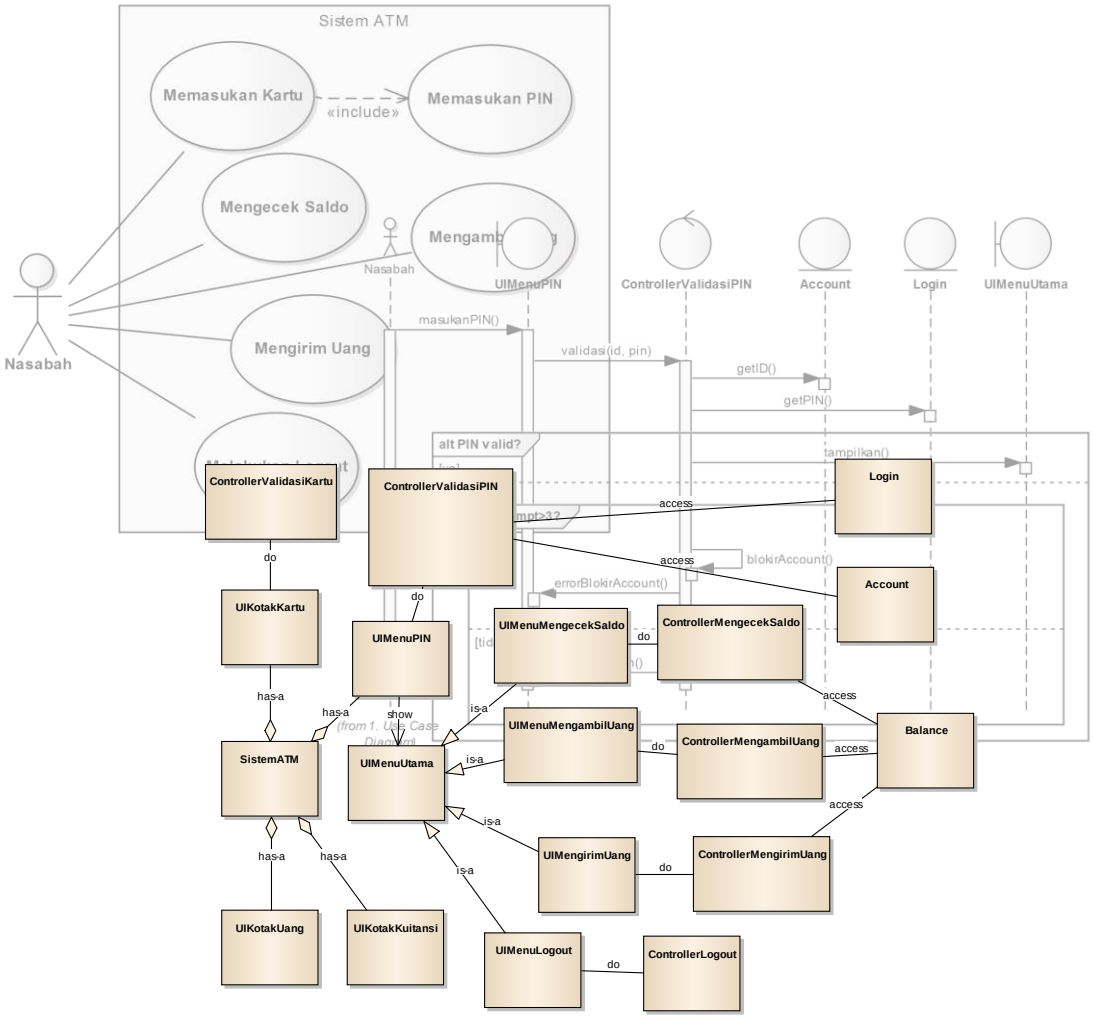
Journal of Software Engineering



Editorial Board

Contents

A Systematic Literature Review of Software Defect Prediction: Research Trends, Datasets, Methods and Frameworks

Romi Satria Wahono
Faculty of Computer Science, Dian Nuswantoro University
romi@romisatriawahono.net

Abstract: Recent studies of software defect prediction typically produce datasets, methods and frameworks which allow software engineers to focus on development activities in terms of defect-prone code, thereby improving software quality and making better use of resources. Many software defect prediction datasets, methods and frameworks are published disparate and complex, thus a comprehensive picture of the current state of defect prediction research that exists is missing. This literature review aims to identify and analyze the research trends, datasets, methods and frameworks used in software defect prediction research betweeen 2000 and 2013. Based on the defined inclusion and exclusion criteria, 71 software defect prediction studies published between January 2000 and December 2013 were remained and selected to be investigated further. This literature review has been undertaken as a systematic literature review. Systematic literature review is defined as a process of identifying, assessing, and interpreting all available research evidence with the purpose to provide answers for specific research questions. Analysis of the selected primary studies revealed that current software defect prediction research focuses on five topics and trends: estimation, association, classification, clustering and dataset analysis. The total distribution of defect prediction methods is as follows. 77.46% of the research studies are related to classification methods, 14.08% of the studies focused on estimation methods, and 1.41% of the studies concerned on clustering and association methods. In addition, 64.79% of the research studies used public datasets and 35.21% of the research studies used private datasets. Nineteen different methods have been applied to predict software defects. From the nineteen methods, seven most applied methods in software defect prediction are identified. Researchers proposed some techniques for improving the accuracy of machine learning classifier for software defect prediction by ensembling some machine learning methods, by using boosting algorithm, by adding feature selection and by using parameter optimization for some classifiers. The results of this research also identified three frameworks that are highly cited and therefore influential in the software defect prediction field. They are Menzies et al. Framework, Lessmann et al. Framework, and Song et al. Framework.

Keywords: systematic literature review, software defect prediction, software defect prediction methods, NASA MDP datasets

1 INTRODUCTION

A software defect is a fault, error, or failure in a software (Naik and Tripathy 2008). It produces either an incorrect, or unexpected result, and behaves in unintended ways. It is a deficiency in a software product that causes it to perform unexpectedly (McDonald, Musson, & Smith, 2007). The definition of a defect is also best described by using the standard IEEE definitions of error, defect and failure (IEEE, 1990). An error is an action taken by a developer that results in a defect. A defect is the manifestation of an error in the code whereas a failure is the incorrect behavior of the system during execution. A developer error can also be defined as a mistake.

As today's software grows rapidly in size and complexity, software reviews and testing play a crucial role in the software development process, especially in capturing software defects. Unfortunately, software defects or software faults are very expensive in cost. Jones and Bonsignour (2012) reported that the cost of finding and correcting defects is one of the most expensive software development activities (Jones and Bonsignour 2012). The cost of software defect increases over the software development step. During the coding step, capturing and correcting defects costs $977 per defect. The cost increases to $7,136 per defect in the software testing phase. Then in the maintenance phase, the cost to capture and remove increases to $14,102 (Boehm and Basili 2001).

Software defect prediction approaches are much more cost-effective to detect software defects as compared to software testing and reviews. Recent studies report that the probability of detection of software defect prediction models may be higher than probability of detection of currently software reviews used in industrial methods (Menzies et al., 2010). Therefore, accurate prediction of defect-prone software helps to direct test effort, to reduce costs, to improve the software testing process by focusing on defect-prone modules (Catal, 2011), and finally to improve the quality of the software (T. Hall, Beecham, Bowes, Gray, & Counsell, 2012). That is why, today software defect prediction is a significant research topic in the software engineering field (Song, Jia, Shepperd, Ying, & Liu, 2011).

Many software defect prediction datasets, methods and frameworks are published disparate and complex, thus a comprehensive picture of the current state of defect prediction research that exists is missing. This literature review aims to identify and analyze the research trends, datasets, methods and frameworks used in software defect prediction research betweeen 2000 and 2013.

This paper is organized as follows. In section 2, the research methodology are explained. The results and answers of research questions are presented in section 3. Finally, our work of this paper is summarized in the last section.

2 METHODOLOGY

2.1 Review Method

A systematic approach for reviewing the literature on the software defect prediction is chosen. Systematic literature reviews (SLR) is now a well established review method in software engineering. An SLR is defined as a process of

identifying, assessing, and interpreting all available research evidence with the purpose to provide answers for specific research questions (Kitchenham and Charters 2007). This literature review has been undertaken as a systematic literature review based on the original guidelines proposed by Kitchenham and Charters (2007). The review method, style and some of the figures in this section were also motivated by (Unterkalmsteiner et al., 2012) and (Radjenović, Heričko, Torkar, & Živkovič, 2013).

As shown in Figure 1, SLR is performed in three stages: planning, conducting and reporting the literature review. In the first step the requirements for a systematic review are identified (Step 1). The objectives for performing the literature review were discussed in the introduction of this chapter. Then, the existing systematic reviews on software defect prediction are identified and reviewed. The review protocol was designed to direct the execution of the review and reduce the possibility of researcher bias (Step 2). It defined the research questions, search strategy, study selection process with inclusion and exclusion criteria, quality assessment, and finally data extraction and synthesis process. The review protocol is presented in Sections 2.2, 2.3, 2.4 and 2.5. The review protocol was developed, evaluated and iteratively improved during the conducting and reporting stage of the review.

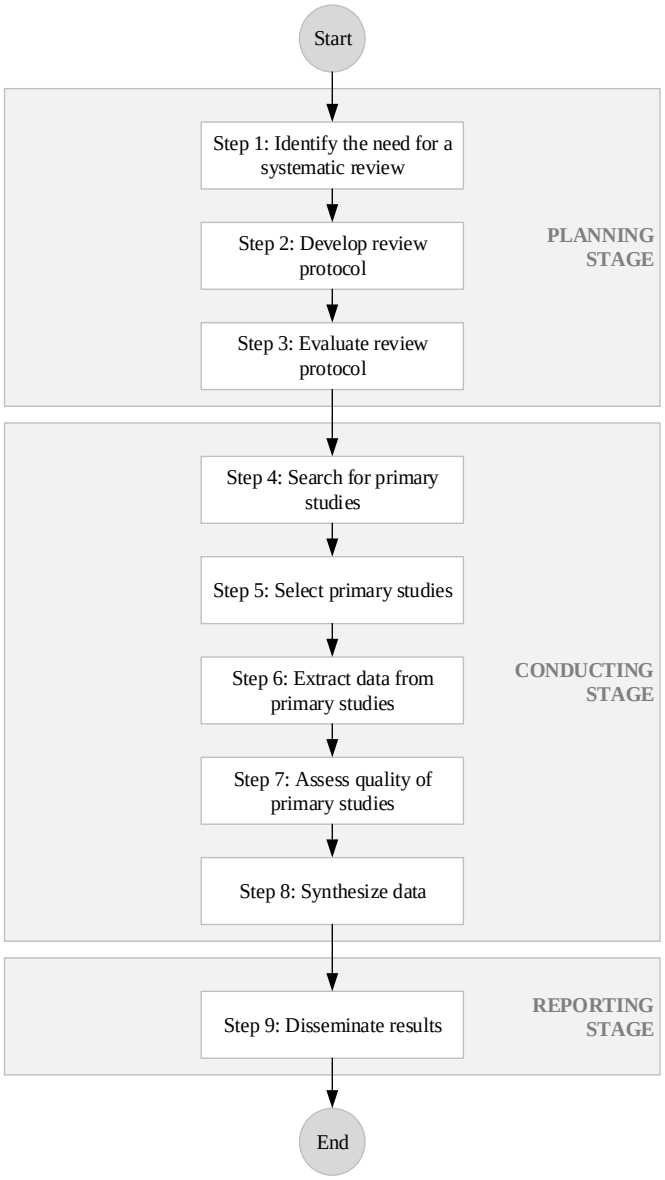


Figure 1 Systematic Literature Review Steps

2.2 Research Questions

The research questions (RQ) were specified to keep the review focused. They were designed with the help of the Population, Intervention, Comparison, Outcomes, and Context (PICOC) criteria (Kitchenham and Charters 2007). Table 1 shows the (PICOC) structure of the research questions.

Table 1 Summary of PICOC

Population	Software, software application, software system, information system
Intervention	Software defect prediction, fault prediction, error-prone, detection, classification, estimation, models, methods, techniques, datasets
Comparison	n/a
Outcomes	Prediction accuracy of software defect, successful defect prediction methods
Context	Studies in industry and academia, small and large data sets

The research questions and motivation addressed by this literature review are shown in Table 2.

Table 2 Research Questions on Literature Review

ID	Research Question	Motivation
RQ1	Which journal is the most significant software defect prediction journal?	Identify the most significant journals in the software defect prediction field
RQ2	Who are the most active and influential researchers in the software defect prediction field?	Identify the most active and influential researchers who contributed so much on a research area of software defect prediction
RQ3	What kind of research topics are selected by researchers in the software defect prediction field?	Identify research topics and trends in software defect prediction
RQ4	What kind of datasets are the most used for software defect prediction?	Identify datasets commonly used in software fault prediction
RQ5	What kind of methods are used for software defect prediction?	Identify opportunities and trends for software defect prediction method
RQ6	What kind of methods are used most often for software defect prediction?	Identify the most used methods for software defect prediction
RQ7	Which method performs best when used for software defect prediction?	Identify the best method in software defect prediction
RQ8	What kind of method improvements are proposed for software defect prediction?	Identify the proposed method improvements for predicting the software defect
RQ9	What kind of frameworks are proposed for software defect prediction?	Identify the most used frameworks in software defect prediction

From the primary studies, software prediction methods, frameworks and datasets to answer RQ4 to RQ9 are extracted. Then, the software defect prediction methods, frameworks and datasets were analyzed to determine which ones are, and which are not, significant methods, frameworks and datasets in software defect prediction (RQ4 to RQ9). RQ4 to RQ9 are the main research questions, and the remaining questions (RQ1 to RQ3) help us evaluate the context of the primary studies. RQ1 to RQ3 give us a summary and synopsis of a particular area of research in software defect prediction field.

Figure 2 shows the basic mind map of the systematic literature review. The main objective of this systematic literature review is to identify software prediction methods, framework and datasets used in software defect prediction.

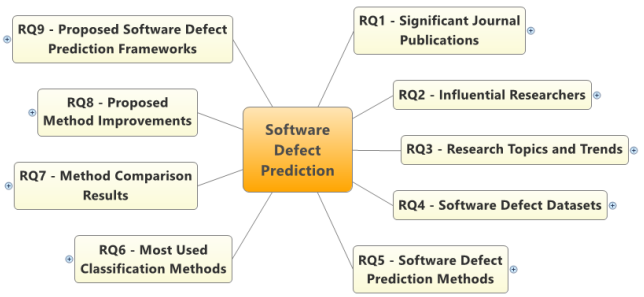


Figure 2 Basic Mind Map of the SLR on Software Defect Prediction

2.3 Search Strategy

The search process (Step 4) consists of some activities, such as selecting digital libraries, defining the search string, executing a pilot search, refining the search string and retrieving an initial list of primary studies from digital libraries matching the search string. Before starting the search, an appropriate set of databases must be chosen to increase the probability of finding highly relevant articles. The most popular literature databases in the field are searched to have the broadest set of studies possible. A broad perspective is necessary for an extensive and broad coverage of the literature. Here is the list of the digital databases searched:

- ACM Digital Library (*dl.acm.org*)
- IEEE eXplore (*ieeexplore.ieee.org*)
- ScienceDirect (*sciencedirect.com*)
- Springer (*springerlink.com*)
- Scopus (*scopus.com*)

The search string was developed according to the following steps:

1. Identification of the search terms from PICOC, especially from Population and Intervention
2. Identification of search terms from research questions
3. Identification of search terms in relevant titles, abstracts and keywords
4. Identification of synonyms, alternative spellings and antonyms of search terms
5. Construction of sophisticated search string using identified search search terms, Boolean ANDs and ORs

The following search string was eventually used:

(software OR applicati OR systems) AND (fault* OR defect* OR quality OR error-prone) AND (predict* OR prone* OR probability OR assess* OR detect* OR estimat* OR classificat*)*

The adjustment of the search string was conducted, but the original one was kept, since the adjustment of the search string would dramatically increase the already extensive list of irrelevant studies. The search string was subsequently adjusted to suit the specific requirements of each database. The databases were searched by title, keyword and abstract. The search was limited by the year of publication: 2000-2013. Two kinds of publication namely journal papers and conference proceedings were included. The search was limited only articles published in English.

2.4 Study Selection

The inclusion and exclusion criteria were used for selecting the primary studies,. These criteria are shown in Table 3.

Table 3 Inclusion and Exclusion Criteria

Inclusion Criteria	Studies in academic and industry using large and small scale data sets
	Studies discussing and comparing modeling performance in the area of software defect prediction
	For studies that have both the conference and journal versions, only the journal version will be included
	For duplicate publications of the same study, only the most complete and newest one will be included
Exclusion Criteria	Studies without a strong validation or including experimental results of software defect prediction
	Studies discussing defect prediction datasets, methods, frameworks in a context other than software defect prediction
	Studies not written in English

Software package Mendeley (*http://mendeley.com*) was used to store and manage the search results. The detailed search process and the number of studies identified at each phase are shown in Figure 3. As shown in Figure 3, the study selection process (Step 5) was conducted in two steps: the exclusion of primary studies based on the title and abstract and the exclusion of primary studies based on the full text. The literature review studies and other studies which do not include experimental results are excluded. The similarity degree of the study with software defect prediction is also the inclusion of studies.

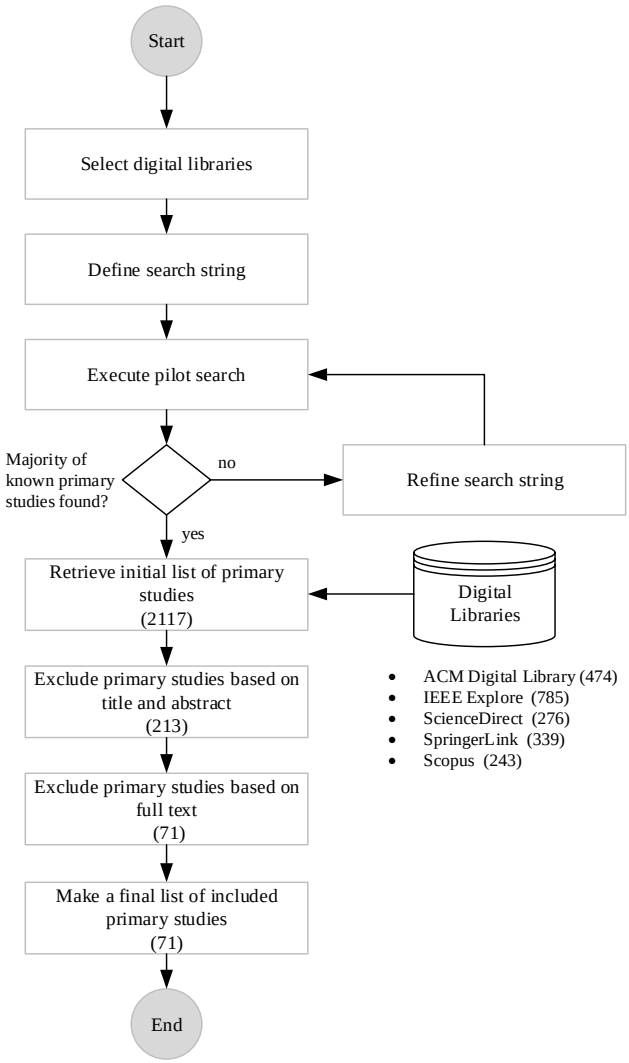


Figure 3 Search and Selection of Primary Studies

The final list of selected primary studies for the first stage had 71 primary studies. Then, the full texts of 71 primary studies were analyzed. In addition to the inclusion and exclusion criteria, the quality of the primary studies, their relevance to the research questions and study similarity were considered. Similar studies by the same authors in various journals were removed. 71 primary studies remained after the exclusion of studies based on the full text selection. The complete list of selected studies is provided in last section section of this paper (Table 6).

2.5 Data Extraction

The selected primary studies are extracted to collect the data that contribute to addressing the research questions concerned in this review. For each of the 71 selected primary studies, the data extraction form was completed (Step 6). The data extraction form was designed to collect data from the primary studies needed to answer the research questions. The properties were identified through the research questions and analysis we wished to introduce. Six properties were used to answer the research questions shown in Table 4. The data extraction is performed in an iterative manner.

Table 4 Data Extraction Properties Mapped to Research Questions

Property	Research Questions
Researchers and Publications	RQ1, RQ2
Research Trends and Topics	RQ3
Software Defect Datasets	RQ4
Software Metrics	RQ4
Software Defect Prediction Methods	RQ5, RQ6, RQ7, RQ8
Software Defect Prediction Frameworks	RQ9

2.6 Study Quality Assessment and Data Synthesis

The study quality assessment (Step 8) can be used to guide the interpretation of the synthesis findings and to define the strength of the elaborated inferences. The goal of data synthesis is to aggregate evidence from the selected studies for answering the research questions. A single piece of evidence might have small evidence force, but the aggregation of many of them can make a point stronger. The data extracted in this review include both quantitative data and qualitative data. Different strategies were employed to synthesize the extracted data pertaining to different kinds of research questions. Generally, the narrative synthesis method was used. The data were tabulated in a manner consistent with the questions. Some visualization tools, including bar charts, pie charts, and tables were also used to enhance the presentation of the distribution of software defect prediction methods and their accuracy data.

2.7 Threats to Validity

This review aims to analyze the studies on software defect prediction based on statistical and machine learning techniques. This review is not aware about the existence of biases in choosing the studies. The searching was not based on manual reading of titles of all published papers in journals. This means that this review may have excluded some software defect prediction papers from some conference proceedings or journals.

This review did not exclude studies from conference proceedings because experience reports are mostly published in conference proceedings. Therefore, a source of information about the industry's experience is included. Some systematic literature reviews, for example (Jorgensen and Shepperd 2007) did not use conference proceedings in their review because workload would increase significantly. A systematic literature review that included studies in conference proceedings as the primary studies is conducted by Catal and Diri (Catal and Diri 2009a).

3 RESEARCH RESULTS

3.1 Significant Journal Publications

In this literature review, 71 primary studies that analyze the performance of software defect prediction are included. The distribution over the years is presented to show how the interest in software defect prediction has changed over time. A short overview of the distribution studies over the years is shown in Figure 4. More studies were published since 2005, indicating that more contemporary and relevant studies are included. It should be noted that the PROMISE repository was developed in 2005, and researchers began to be aware of the use of public datasets. Figure 4 also shows that the research field on software defect prediction is still very much relevant today.

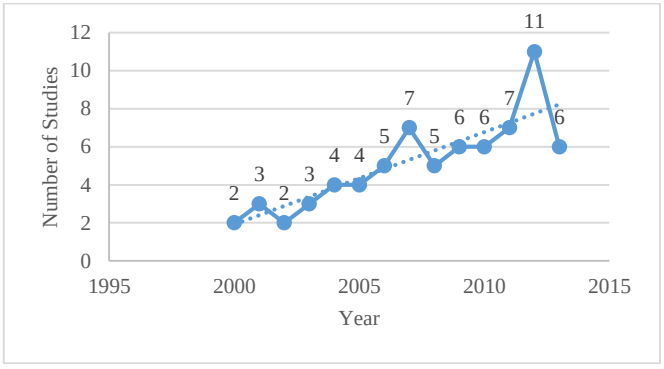


Figure 4 Distribution of Selected Studies over the Years

According to the selected primary studies, the most important software defect prediction journals are displayed in Figure 5. Note that the conference proceedings are not included in this graph.

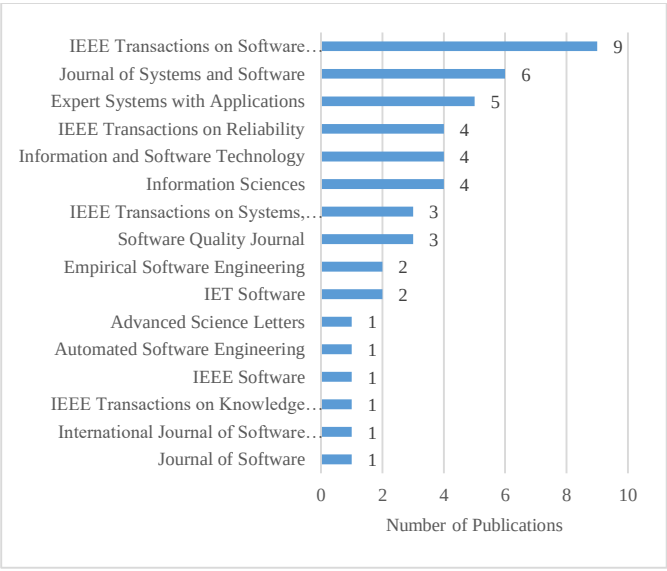


Figure 5 Journal Publications and Distribution of Selected Studies

Table 5 shows the Scimago Journal Rank (SJR) value and Q categories (Q1-Q4) of the most important software defect prediction journals. Journal publications are ordered according to their SJR value.

Table 5 Scimago Journal Rank (SJR) of Selected Journals

No	Journal Publications	SJR	Q Category
1	IEEE Transactions on Software Engineering	3.39	Q1 in Software
2	Information Sciences	2.96	Q1 in Information Systems
3	IEEE Transactions on Systems, Man, and Cybernetics	2.76	Q1 in Artificial Intelligence
4	IEEE Transactions on Knowledge and Data Engineering	2.68	Q1 in Information Systems
5	Empirical Software Engineering	2.32	Q1 in Software
6	Information and Software Technology	1.95	Q1 in Information Systems
7	Automated Software Engineering	1.78	Q1 in Software
8	IEEE Transactions on Reliability	1.43	Q1 in Software
9	Expert Systems with Applications	1.36	Q2 in Computer Science
10	Journal of Systems and Software	1.09	Q2 in Software
11	Software Quality Journal	0.83	Q2 in Software
12	IET Software	0.55	Q2 in Software
13	Advanced Science Letters	0.24	Q3 in Computer Science
14	Journal of Software	0.23	Q3 in Software
15	International Journal of Software Engineering and Its Application	0.14	Q4 in Software

3.2 Most Active and Influential Researchers

From the selected primary studies, researchers who contributed very well and who are very active in the software defect prediction research field were investigated and identified. Figure 6 shows the most active and influential researchers in the software defect prediction field. The researchers were listed according to the number of studies included in the primary studies. It should be noted that Taghi Khoshgoftaar, Tim Menzies, Qinbao Song, Martin Shepperd, Norman Fenton, Cagatay Catal, Burak Turhan, Ayse Bener, Huanjing Wang, Yan Ma, Bojan Cukic, and Ping Guo are active researchers on software defect prediction.

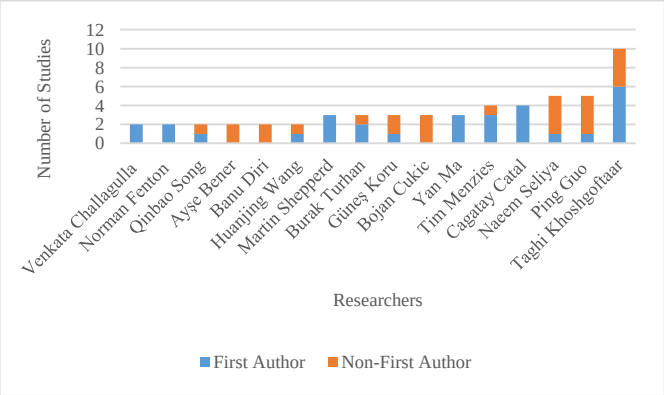


Figure 6 Influential Researchers and Number of Studies

3.3 Research Topics in the Software Defect Prediction Field

Software defect prediction is a significant research topic in the software engineering field (Song et al., 2011). Analysis of the selected primary studies revealed that current software defect prediction research focuses on five topics:

1. Estimating the number of defects remaining in software systems, using the estimation algorithm (**Estimation**)
2. Discovering defect associations using the association rule algorithm (**Association**)
3. Classifying the defect-proneness of software modules typically into two classes namely defect-prone and not defect-prone using the classification algorithm (**Classification**)
4. Clustering the software defect based on object using the clustering algorithm (**Clustering**)
5. Analyzing and pre-processing the software defect datasets (**Dataset Analysis**)

The first type of work (**Estimation**) applies statistical approaches (Ostrand, Weyuker, & Bell, 2005), capture-recapture models (Emam and Laitenberger 2001), and neural network (Benaddy and Wakrim 2012) (Zhang and Chang 2012) to estimate the number of defects remaining in softwares with inspection data and process quality data. The prediction result can be used as an important tool to help software developers (Kenny, 1993), and can be used to control the software process and gauge the likely delivered quality of a software system (Fenton and Neil 1999).

The second type of work (**Association**) uses association rule mining algorithms from the data mining community to expose software defect associations (Shepperd, Cartwright, & Mair, 2006) (Karthik and Manikandan 2010) (C.-P. Chang, Chu, & Yeh, 2009). This second type of work can be used for three purposes (Song et al., 2011). Firstly, to find as many related defects as possible to the captured defects and consequently, make more effective improvements to the software. This may be useful as it permits more focused testing and more effective use of limited testing resources. Secondly, to evaluate the results from software reviewers during an inspection. Thus, the work should be reinspected for completeness. Thirdly, to assist software development managers in improving the software development process through analysis of the reasons why some defects frequently occur together. Managers can than devise corrective action, if the analysis leads to the identification of a process problem.

The third type of work (**Classification**) classifies software modules as defect-prone and non-defect-prone by means of metric based classification (Khoshgoftaar *et al.* 2000) (Li and Reformat 2007) (Cukic and Singh 2004) (Menzies, Greenwald, & Frank, 2007) (Lessmann, Baesens, Mues, & Pietsch, 2008) (Song et al., 2011). The classification algorithm is a popular machine learning approach for software defect prediction (Lessmann et al., 2008). It categorizes the software code attributes into defective or not defective, which is completed by means of a classification model derived from software metrics data based on the previous development projects (Gayatri, Reddy, & Nickolas, 2010). The classification algorithm is able to predict which components are more likely to be defect-prone which supports a better targeted testing resources. If an error is reported during system tests or from field tests, that module's fault data is marked as 1, otherwise 0. For prediction modeling, software metrics are used as independent variables and fault data is used as the dependent variable (Catal, 2011). Parameters of the prediction model are computed by using previous software metrics and fault data. Various types of classification algorithms have been applied for software defect prediction (Lessmann et al., 2008), including logistic regression (Denaro, 2000), decision trees (Khoshgoftaar and Seliya, 2002) (Taghi M Khoshgoftaar, Seliya, & Gao, 2005), neural networks (Park, Oh, & Pedrycz, 2013) (Wang and Yu 2004) (Zheng, 2010), and naive bayes (Menzies et al., 2007).

The fourth type of work (**Clustering**) uses clustering algorithms from the data mining community to capture software defect clusters. Unsupervised learning methods like clustering may be used for defect prediction in software modules, more so in those cases where fault labels are not available. The K-Means algorithm was proposed by Bishnu

and Bhattacherjee (2012) for predicting defect in program modules (Bishnu and Bhattacherjee 2012). Quad Trees are applied for finding the initial cluster centers to be the input to the K-Means Algorithm. The concept of clustering gain has been used to define the quality of clusters for measuring the Quad Tree-based initialization algorithm. The clusters generated by the Quad Tree-based algorithm were found to have maximum gain values (Bishnu and Bhattacherjee 2012).

The fifth type of work (**Dataset Analysis**) focuses on analyzing and pre-processing the software defect datasets. Some researchers conducted the dataset pre-processing using some methods, while others analyzed software defect datasets in multiple aspect of views. (Gray, Bowes, Davey, Sun, & Christianson, 2012) demonstrated and explained why NASA MDP datasets require significant pre-processing in order to be suitable for defect prediction. They noted that the bulk of defect prediction experiments based on the NASA Metrics Data Program datasets may have led to erroneous findings. This is mainly due to repeated data points potentially caused by redundancy in the amount of training and testing data.

Figure 7 shows the total distribution of research topics on software defect prediction from 2000 until 2013. 77.46% of the research studies are related to classification topics, 14.08% of the studies focused on estimation techniques, and 5.63% of the primary studies are concerned with dataset analysis topics. Clustering and association are minor research topics with only 1.41% coverage. It can be concluded that most of the software defect prediction researchers selected classification as their research topics. There are three possible reasons of why researchers focus on this topic. As the first reason, classification topics precisely match with the industrial needs that require some methods to predict which modules are more likely to be defect-prone. Thus, the result of prediction can be used to support better targeted testing resources. The second reason is related to the NASA MDP dataset that is mostly ready for classification methods. The third possible reason for a lack of studies in clustering and association related topics is that clustering and association methods usually yield undesirable performance which cannot be published in the literature.

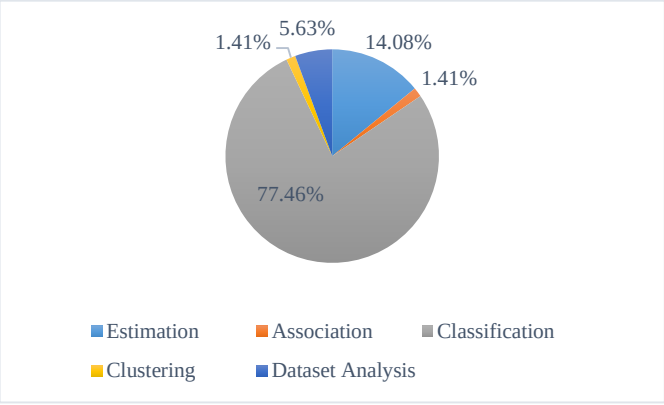


Figure 7 Distribution of Research Topics

3.4 Datasets Used for Software Defect Prediction

A dataset is a collection of data used for some specific machine learning purpose (Sammut and Webb 2011). A training set is a data set that is used as input to a learning system, which analyzes it to learn a model. A test set or evaluation set is a data set containing data that are used to evaluate the model learned by a learning system. A training set may be further divided into a growing set and a pruning set, where the training set and the test set that contain disjoint sets of data, the test set is known as a holdout set.

One of the most critical problems for software defect prediction studies is the usage of non-public datasets (Catal and Diri 2009a). Numerous companies developed defect prediction models using proprietary data and presented these models in conferences. However, it is impossible to compare results of such studies with results of the proposed models, because their datasets cannot be assesed. Machine learning researchers had similar problems in the 1990s, and they developed a repository called University of California Irvine (UCI). Inspired by the UCI effort, software engineering researchers developed the PROMISE repository which has numerous public datasets in 2005. NASA software defect prediction datasets are located in PROMISE. The ARFF format is used as a default format file that makes it possible to use these datasets directly from WEKA or RapidMiner, an open source machine learning software.

In this literature review, 71 primary studies that analyzed the performance of software defect prediction are included. Figure 8 shows the distribution of dataset types from 2000 until 2013. 64.79% of the research studies used public datasets and 35.21% of the research studies used private datasets. Public datasets are mostly located in the PROMISE and NASA MDP (metrics data program) repositories and they are distributed freely. Private datasets belong to private companies and they are not distributed as public datasets.

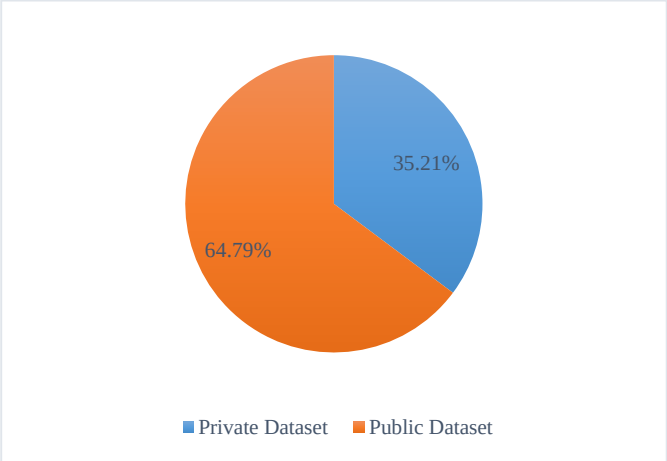


Figure 8 Total Distribution of Datasets

The distribution over the years is presented to show how the interest in dataset types has changed over time. Unfortunately, totally 35.21% of the studies used private datasets. This means that only the result of one study from three studies can be compared and it is repeatable. However, it is not possible to compare the results of such studies with the results of the proposed models because their datasets are not distributed as public. The use of standard datasets make the research repeatable, refutable, and verifiable (Catal and Diri 2009a). The distribution of the primary studies over the years, and per source, is presented in Figure 9. More studies have been published, and more public datasets have been used for the software defect prediction research since 2005. As mentioned earlier, the PROMISE repository was developed in 2005. In addition, there is increased awareness among researchers on the use of public datasets.

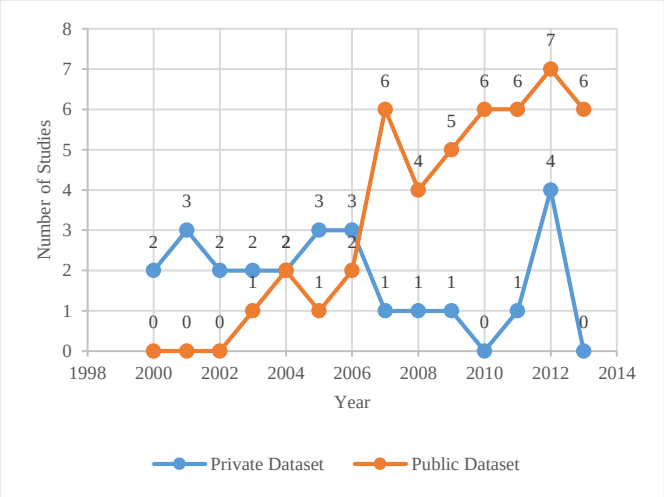


Figure 9 Distribution of Private and Public Datasets

3.5 Methods Used in Software Defect Prediction

As shown in Figure 10, since 2000, nineteen methods have been applied and proposed as the best method to predict software defects. A summary of the state-of-the-art methods used in software defect prediction is shown in Figure 10 and Table 6.

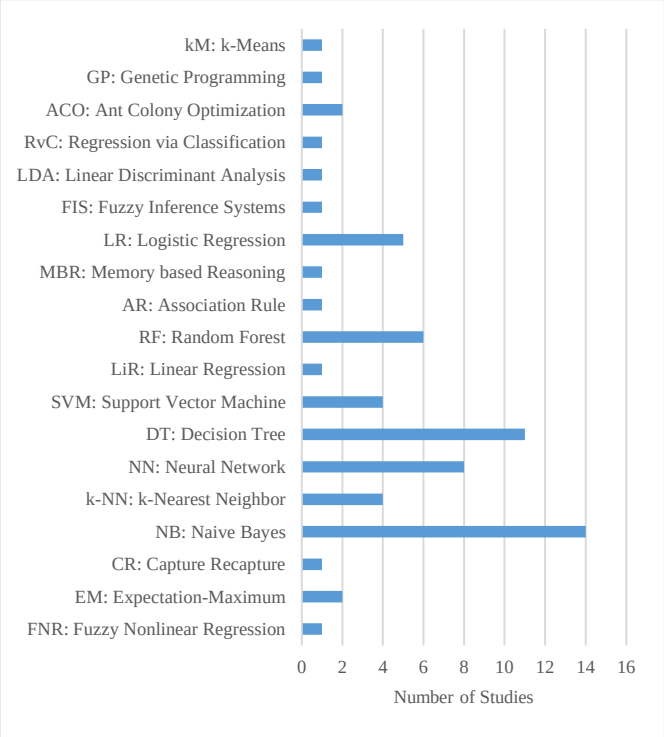


Figure 10 Methods Used in Software Defect Prediction

3.6 Most Used Methods in Software Defect Prediction

From the nineteen methods shown in Figure 10 in Section 3.5, seven most applied classification methods in software defect prediction are identified. The methods are shown in Figure 11. They are:

1. Logistic Regression (LR)
2. Naïve Bayes (NB)
3. K-Nearest Neighbor (k-NN)
4. Neural Network (NN)
5. Decision Tree (DT)
6. Support Vector Machine (SVM)
7. Random Forest (RF)

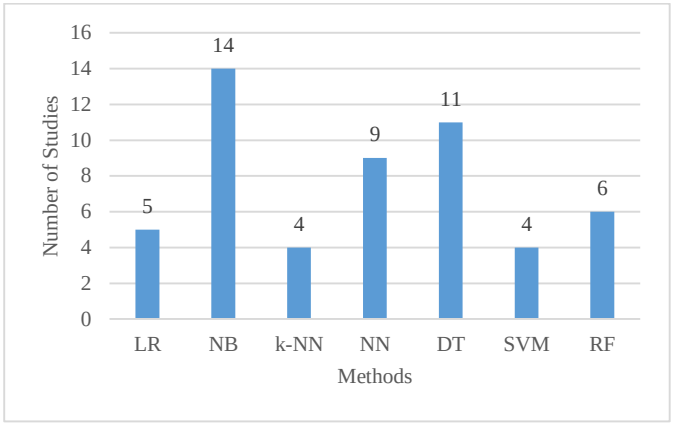


Figure 11 Most Used Methods in Software Defect Prediction

NB, DT, NN and RF are the four most frequently used ones. They were adopted by 75% of the selected studies, as illustrated in Figure 12.

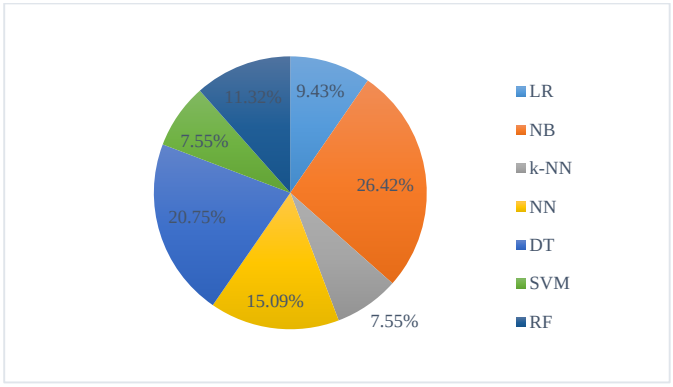


Figure 12 Distribution of the Studies over Type of Methods

3.7 Method Perform Best for Software Defect Prediction

While many studies in the software defect prediction individually report the comparative performance of the modelling techniques used, there is no strong consensus on which performs best when the studies are looked at individual. Bibi *et al.* (Bibi, Tsoumakas, Stamelos, & Vlahavas, 2008) have reported that Regression via Classification (RvC) works very well. Hall *et al.* highlighted that studies using Support Vector Machine (SVM) perform less well. These may be performing bellow expectation as they require parameter optimization for the best performance (T. Hall et al., 2012). C4.5 seems to perform bellow expectation if they include imbalanced class distribution of datasets, as the algorithm seems to be sensitive to this (Arisholm, Briand, & Fuglerud, 2007) (Arisholm, Briand, & Johannessen, 2010).

Naïve Bayes (NB) and Logistic Regression (LR) seem to be the methods used in models that performs relatively well in the field of software defect prediction (Menzies et al., 2007) (Song et al., 2011). NB is a well understood algorithm and commonly in use. Studies using Random Forests (RF) did not perform as well as expected (T. Hall et al., 2012). However, many studies using the NASA dataset employ RF and report good performanc (Lessmann et al., 2008).

Some studies on software defect prediction indicated that Neural Network (NN) has a good accuracy as a classifier (Lessmann et al., 2008) (Benaddy and Wakrim 2012) (Quah, Mie, Thwin, & Quah, 2003) (T M Khoshgoftaar, Allen, Hudepohl, & Aud, 1997). NN has been shown to be more adequate for the problem on the complicated and nonlinear relationship between software metrics and defect-proneness of

software modules (Zheng 2010). However, the practicability of NN is limited due to difficulty in selecting appropriate parameters of network architecture, including number of hidden neuron, learning rate, momentum and training cycles (Lessmann et al., 2008).

However, models seem to have performed best where the right technique has been selected for the right set of data. No particular classifiers that performs the best for all the datasets (Challagulla, Bastani, and Paul, 2005) (Song et al., 2011). Therefore, the comparisons and benchmarking results of defect prediction using machine learning classifiers indicate that the poor accuracy level is dominant (Sandhu, Kumar, & Singh, 2007) (Lessmann et al., 2008), significant performance differences could not be detected (Lessmann et al., 2008) and no particular classifiers perform the best for all the datasets (Challagulla, Bastani, and Paul, 2005) (Song et al., 2011).

3.8 Proposed Method Improvements for Software Defect Prediction

Researchers proposed some techniques for improving the accuracy of machine learning classifier for software defect prediction. Recent proposed techniques try to increase the prediction accuracy of a generated model by: 1) modifying and ensembling some machine learning methods (Mısırlı, Bener, & Turhan, 2011) (Tosun, Turhan, & Bener, 2008), 2) using boosting algorithm (Zheng, 2010) (Jiang, Li, Zhou, & Member, 2011), 3) adding feature selection (Gayatri *et al.* 2010) (Khoshgoftaar and Gao, 2009) (Catal and Diri 2009b) (Song et al., 2011), 4) by using parameter optimization for some classifiers (Peng and Wang 2010) (Lin, Ying, Chen, & Lee, 2008) (X. C. Guo, Yang, Wu, Wang, & Liang, 2008).

However, eventhough various defect prediction methods have been proposed, but none has been proven to be consistently accurate (Challagulla *et al.*, 2005) (Lessmann et al., 2008). The accurate and reliable classification algorithm to build a better prediction model is an open issue in software defect prediction. There is a need for an accurate defect prediction framework which has to be more robust to noise and other problems associated with on datasets.

3.8.1 Feature Selection

Feature selection is the study of algorithms for reducing dimensionality of data to improve machine learning performance. For a dataset with N features and M dimensions (or features, attributes), feature selection aims to reduce M to M' and $M' \leq M$ (Sammut and Webb 2011). It is an important and widely used approach to dimensionality reduction. Another effective approach is feature extraction. One of the key distinctions of the two approaches lies at their outcomes. Assuming we have four features F_1, F_2, F_3, F_4, if both approaches result in 2 features, the 2 selected features are a subset of 4 original features (say, F_1, F_3), but the 2 extracted features are some combination of the 4 original features.

Feature selection is commonly used in applications where original features need to be retained. Some examples are document categorization, medical diagnosis and prognosis as well as gene-expression profiling. The benefits of feature selection are multifold: it helps improve machine learning in terms of predictive accuracy, comprehensibility, learning efficiency, compact models, and effective data collection. The objective of feature selection is to remove irrelevant and/or redundant features and retain only relevant features (Maimon and Rokach 2010). Some researchers called irrelevant and redundant feature by noisy attribute (Khoshgoftaar and Van Hulse 2009). Irrelevant features can be removed without affecting learning performance. Redundant features are a type of irrelevant features. The distinction is that a redundant feature implies the copresence of another feature; individually, each feature is relevant, but the removal of either one will not affect learning performance.

Three classic methods of feature selection are filter, wrapper, and embedded. Research shows that a classifier with embedded feature selection capability can beneft from feature selection in terms of learning performance. A filter model relies on measures about the intrinsic data properties. Mutual information and data consistency are two examples of measures about data properties. A wrapper model involves a learning algorithm (classifier) in determining the feature quality. For instance, if removing a feature does not affect the classifier's accuracy, the feature can be removed. Obviously, this way feature selection is adapted to improving a particular classification algorithm. To determine if the feature should be selected or removed, it needs to build a classifier every time when a feature is considered. Hence, the wrapper model can be quite costly. An embedded model embeds feature selection in the learning of a classifier. The best example can be found in decision tree induction in which a feature has to be selected first at each brainching point. When feature selection is performed for data preprocessing, fillter and wrapper models are often employed. When the purpose of feature selection goes beyond improving learning performance (e.g., classifcation accuracy), the most applied is the filter model.

3.8.2 Ensemble Machine Learning

Ensemble learning refers to the procedures employed to train multiple learning machines and combine their outputs, treating them as a "committee" of decision makers (Sammut and Webb 2011). The principle is that the decision of the committee, with individual predictions combined appropriately, should have better overall accuracy, on average, than any individual committee member. Numerous empirical and theoretical studies have demonstrated that ensemble models very often attain higher accuracy than single models.

The members of the ensemble might be predicting real-valued numbers, class labels, posterior probabilities, rankings, clusterings, or any other quantity. Therefore, their decisions can be combined by many methods, including averaging, voting, and probabilistic methods. The majority of ensemble learning methods are generic as well as applicable across broad classes of model types and learning tasks.

Several machine learning techniques do this by learning an ensemble of models and using them in combination. Prominent among these are schemes called bagging, boosting, and stacking (Witten, Frank, & Hall, 2011). They can all, more often than not, increase predictive performance over a single model. They are general techniques that can be applied to classification tasks and numeric prediction problems. Bagging, boosting, and stacking have been developed over the last couple of decades, and their performance is often astonishingly good. Machine learning researchers have struggled to understand why. And during that struggle, new methods have emerged that are sometimes even better. For example, while human committees rarely benefit from noisy distractions, shaking up bagging by adding random variants of classifiers can improve performance.

3.9 Proposed Frameworks for Software Defect Prediction

Three frameworks that are highly cited and therefore influential in the software defect prediction field are the Menzies *et al.* Framework (Menzies et al., 2007), Lessmann *et*

al. Framework (Lessmann et al., 2008), and Song *et al.* Framework (Song et al., 2011).

3.9.1 Menzies *et al.*'s Framework

Menzies *et al.* (2007) published a study which compared the performance of two classification algorithms techniques to predict software components containing defects (Menzies et al., 2007). They used the NASA MDP repository, which contained 10 different datasets. Many researchers have explored issues like the relative merits of Halstead's software science measures, McCabe's cyclomatic complexity and lines of code counts for building defect predictors. However, Menzies *et al.* (2007) claim that such debates are irrelevant since how the attributes are used to build predictors is much more important than which particular attributes are used, and the choice of learning method is far more important than which subset of the available data is used for learning (Menzies et al., 2007). Their research revealed that a Naive Bayes classifier had a mean probability of detection of 71 percent and mean false alarms rates of 25 percent, after log filtering and attribute selection based on InfoGain. Naive bayes significantly outperformed the rule induction methods of J48 and OneR. However, the choice of which attribute subset is used for learning is not only circumscribed by the attribute subset itself and available data, but also by attribute selectors, learning algorithms, and data preprocessors. An intrinsic relationship between a learning method and an attribute selection method is well known. For example, Hall and Holmes (2003) concluded that the backward elimination (BE) search is more suitable for C4.5, but the forward selection (FS) search was well suited to Naive Bayes (Hall and Holmes 2003). Therefore, Menzies *et al.* chose the combination of all learning algorithm, data preprocessing, and attribute selection method before building prediction models. Figure 13 shows Menzies *et al.*'s software defect prediction framework.

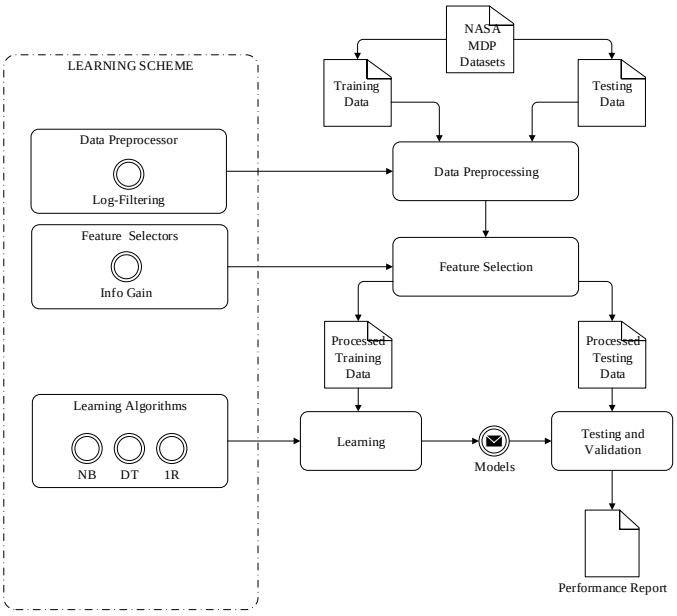


Figure 13 Menzies *et al.*'s Framework
(Compiled from (Menzies et al., 2007))

3.9.2 Lessmann *et al.*'s Framework

Lessmann *et al.* also conducted a follow up to Menzies *et al.*'s framework on defect predictions (Lessmann et al., 2008). However, Lessmann *et al.* did not perform attribute selection when building prediction models. Lessmann *et al.* consider three potential sources for bias: 1) relying on accuracy indicators that are conceptually inappropriate for software defect prediction and cross-study comparisons, 2) limiting use of statistical testing procedures to secure empirical findings, and 3) comparing classifiers over one or a small number of proprietary datasets. Lessman *et al.* (2008) proposed a framework for comparative software defect prediction experiments. This framework is implemented on a large scale empirical comparison of 22 classifiers over 10 datasets from the NASA Metrics Data repository. An appealing degree of predictive accuracy is observed, which supports the view that the metric based classification is useful. However, the results showed that no significant performance differences could be detected among the top 17 classifiers. It indicates that the importance of the particular classification algorithm may be less than previously assumed. Figure 14 shows Lessman *et al.*'s software defect prediction framework.

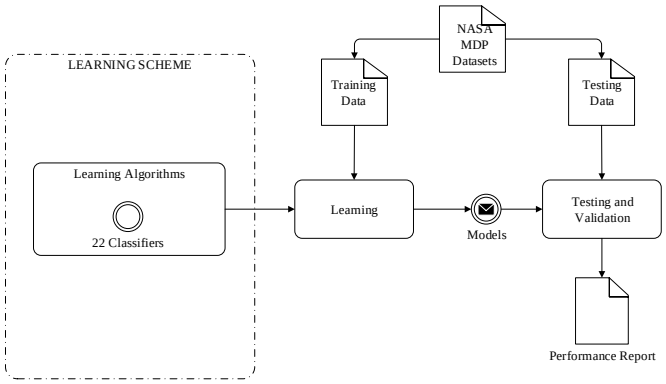


Figure 14 Lessmann *et al.*'s Framework
(Compiled from (Lessmann et al., 2008))

3.9.3 Song *et al.*'s Framework

Song *et al.* (Song et al., 2011) also conducted a follow-up to the results of (Menzies et al., 2007) research on defect predictions. Song *et al.* developed a general-purpose defect prediction framework, which consists of two parts: scheme evaluation and defect prediction. Scheme evaluation focuses on evaluating the performance of a learning scheme, while defect prediction focuses on building a final predictor using historical data according to the learning scheme. Then the predictor is used to predict the defect-prone components of a new software. A learning scheme consists of 1) a data preprocessor, 2) an attribute selector, and 3) a learning algorithm. The main difference between Song *et al.*'s framework and that of Menzies *et al.*'s framework lies in the following. Song *et al.* chose the entire learning scheme, not just one out of the learning algorithm, attribute selector, or data preprocessor.

Song *et al.* also argued that Menzies et al's attribute selection approach is problematic and produced a bias in the evaluation results. One reason is that they ranked attributes on the entire dataset, including both the training and test data, though the class labels of the test data should have been made unknown to the predictor. However, it violated the intention of the holdout strategy. The potential result is that they overestimate the performance of their learning model and thereby report a potentially misleading result. After ranking the attributes, each individual attribute are evaluated separately and the features with the highest scores are chosen. Unfortunately, this approach cannot consider features with complementary information, and does not account for attribute dependence. It is also not capable of eliminating redundant features because redundant features are likely to have similar

rankings. They will all be selected as long as the features are deemed relevant to the class, even though many of them are highly correlated to each other. Figure 15 shows Song *et al.*'s software defect prediction framework.

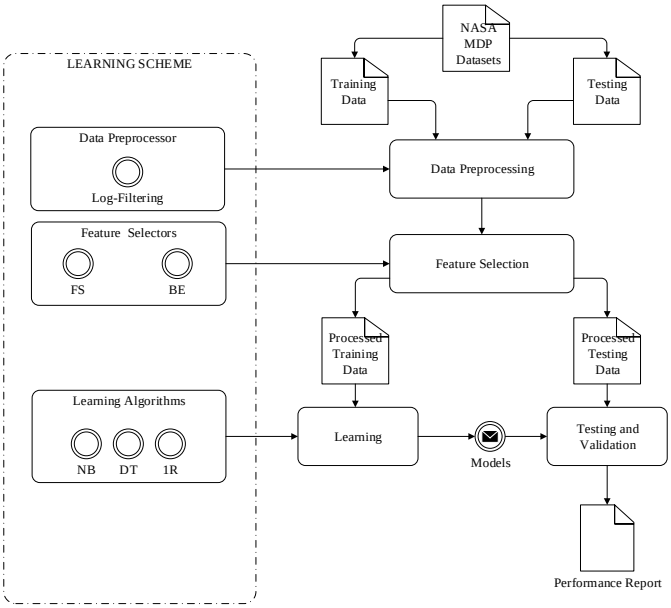


Figure 15 Song *et al.*'s Framework
(Compiled from (Song et al., 2011))

4 CONCLUSION AND FUTURE WORKS

This literature review aims to identify and analyze the trends, datasets, methods and frameworks used in software defect prediction research betweeen 2000 and 2013. Based on the designed inclusion and exclusion criteria, finally 71 software defect prediction studies published between January 2000 and December 2013 were remained and investigated. This literature review has been undertaken as a systematic literature review. Systematic literature review is defined as a process of identifying, assessing, and interpreting all available research evidence with the purpose to provide answers for specific research questions.

Analysis of the selected primary studies revealed that current software defect prediction research focuses on five topics and trends: estimation, association, classification, clustering and dataset analysis. The total distribution of defect prediction methods is as follows. 77.46% of the research studies are related to classification methods, 14.08% of the studies focused on estimation methods, and 1.41% of the studies concerned on clustering and association methods. In addition, 64.79% of the research studies used public datasets and 35.21% of the research studies used private datasets.

Nineteen different methods have been applied to predict software defects. From the nineteen methods, seven most applied methods in software defect prediction are identified. They are Logistic Regression (LR), Naïve Bayes (NB), K-Nearest Neighbor (k-NN), Neural Network (NN), Decision Tree (DT), Support Vector Machine (SVM) and Random Forest (RF)

Researchers proposed some techniques for improving the accuracy of machine learning classifier for software defect prediction by ensembling some machine learning methods, by using boosting algorithm, by adding feature selection and by using parameter optimization for some classifiers.

The results of this research also identified three frameworks that are highly cited and therefore influential in the software defect prediction field. They are the Menzies et al. Framework, Lessmann et al. Framework, and Song et al. Framework.

Unfortunatelly, the existing software defect prediction framework revealed some problems. Unintentionally misleading results and overoptimism on the part of the researchers can result from incomplete validation mechanism. Comprehensive evaluation of different prediction methods is still an open issue in the field of software defect prediction (Mende and Koschke 2009). More reliable research procedures need to be developed, before the confident conclusion of comparative studies of software prediction models can be made (Lessmann et al., 2008) (Myrtveit, Stensrud, & Shepperd, 2005) (Song et al., 2011) (Menzies et al., 2010). This research proposes a new comparison frameworks for software defect prediction in order to fulfill the requirement for more systematic and unbiased methods for comparing the performance of machine-learning-based defect prediction.

Frameworks developed by Menzies *et al.*, Lessmann *et al.*, and Song *et al.* are missing in the processing of class imbalance problem in datasets. Software defect datasets are suferring from an imbalanced problem in datasets with very few defective modules compared to defect-free ones (Wang and Yao 2013) (Zhang and Zhang 2007). The most well-known issue regarding the use of NASA datasets in classification experiments is the variety levels of imbalanced class (Gray *et al.* 2012). Class imbalance either reduces classifier performance (Gray, Bowes, Davey, & Christianson, 2011). The bagging as meta-learning method is used in this study to overcome the class imbalance problem.

The issue of dealing with noisy data has not been addressed adequately in the three frameworks. The noisy and irrelevant features on software defect prediction results in inefficient outcome of the model (Gayatri *et al.* 2010). The software defect prediction accuracy decreases significantly because the dataset contains noisy attributes. The accuracy of software defect prediction improved when irrelevant and redundant attributes are removed. The Lessmann *et al.* framework does not address the issue regarding to the noisy and irrelevant attribute problems. The Menzies *et al.* and Song *et al.* frameworks employed the traditional feature selection algorithms such as information gain, forward selection and backward elimination. In this research, noisy attribute problems were addressed by using metaheuristic optimization methods, especially genetic algorithm and particle swarm optimization. Cano *et al.* (2003) have shown that better results in terms of higher classification accuracy can be obtained with the metaheuristic optimization method than with many traditional and non-evolutionary feature selection methods (Cano, Herrera, & Lozano, 2003).

Finally, the list of primary studies is presented in Table 6. This list is comprised of 6 attributes (year, primary studies, publications, datasets, methods, and topics) and 71 primary studies (from January 2000 to December 2013), and ordered by year of publication.

Figure 16 shows the complete mind map, which presents the results of the systematic literature review on software defect prediction. Mind maps have been used to explore relationships between ideas and elements of an argument and to generate solutions to problems. It puts a new perspective on things to see all the relevant issues and analyze choices in light of the one big picture (Buzan and Griffiths 2013). It also makes it easier to logically organize information and integrate new knowledge. In this research the mind map is used to present the results of the systematic literature review on software defect prediction.

Table 6 The List of Primary Studies in the Field of Software Defect Prediction

Year	Primary Studies	Publications	Datasets	Methods	Topics
2000	(Khoshgoftaar and Allen 2000)	IEEE Transactions on Reliability	Private	Fuzzy Nonlinear Regression	Estimation
	(Lyu, 2000)	Asia-Pacific Conference on Quality Software	Private	Expectation-Maximum	Classification
2001	(Khaled El Emam, Melo, & Machado, 2001)	IEEE Transactions on Software Engineering	Private	Capture-Recapture Model	Estimation
	(N. Fenton, Krause, & Neil, 2001)	IEEE Transactions on Software Engineering	Private	Naïve Bayes	Classification
	(Shepperd and Kadoda 2001)	IEEE Transactions on Software Engineering	Private	k-Nearest Neighbor	Estimation
2002	(Pizzi, Summers, & Pedrycz, 2002)	International Joint Conference on Neural Networks	Private	Neural Network	Classification
	(Khoshgoftaar and Seliya 2002)	IEEE Symposium on Software Metrics	Private	Decision Tree (CART)	Classification
2003	(L. Guo, Cukic, & Singh, 2003)	IEEE Conference on Automated Software Engineering	Public	Neural Network	Classification
	(Quah et al., 2003)	International Conference on Software Maintenance	Private	Neural Network	Estimation
	(Güneş Koru and Tian 2003)	Journal of Systems and Software	Private	Decision Tree	Classification
2004	(Menzies, DiStefano, Orrego, & Chapman, 2004)	IEEE Symposium on High Assurance Systems Engineering	Public	Naïve Bayes	Classification
	(Wang and Yu 2004)	IEEE Conference on Tools with Artificial Intelligence	Private	Neural Network	Classification
	(Kanmani, Uthariaraj, Sankaranarayanan, & Thambidurai, 2004)	ACM SIGSOFT Software Engineering Notes	Private	Neural Network	Estimation
	(V. U. B. Challagulla et al., 2004)	IEEE Workshop on OO Real-Time Dependable Systems	Public	Naïve Bayes	Classification
2005	(Taghi M Khoshgoftaar et al., 2005)	Empirical Software Engineering	Private	Decision Tree	Classification
	(Xing, Guo, & Lyu, 2005)	IEEE Symposium on Software Reliability Engineering	Private	Support Vector Machine	Classification
	(Koru and Liu 2005)	IEEE Software	Public	Decision Tree and Naïve Bayes	Classification
	(Ostrand et al., 2005)	IEEE Transactions on Software Engineering	Private	Linear Regression	Estimation
2006	(Yan Ma, Guo, & Cukic, 2007)	Advances in Machine Learning	Public	Random Forest	Classification
	(Shepperd et al., 2006)	IEEE Transactions on Software Engineering	Public	Association Rule	Association
	(Taghi M. Khoshgoftaar, Seliya, & Sundaresh, 2006)	Software Quality Journal	Private	k-Nearest Neighbor	Estimation
	(V. Challagulla, Bastani, & Yen, 2006)	IEEE Conference on Tools with Artificial Intelligence	Public	Memory based Reasoning	Classification
	(Zhou and Leung 2006)	IEEE Transactions on Software Engineering	Public	Logistic Regression	Classification
2007	(Menzies et al., 2007)	IEEE Transactions on Software Engineering	Public	Naïve Bayes	Classification
	(Li and Reformat 2007)	IEEE Conference on Information Reuse and Integration	Public	Fuzzy Inference System	Classification
	(Yan Ma et al., 2007)	Advances in Machine Learning Applications in Software Engineering	Public	Random Forest	Classification
	(Pai and Dugan 2007)	IEEE Transactions on Software Engineering	Public	Naïve Bayes	Classification
	(Seliya and Khoshgoftaar 2007)	Software Quality Journal	Public	Expectation-Maximum	Classification
	(N. Fenton et al., 2007)	Information and Software Technology	Private	Naïve Bayes	Classification
	(Güneş Koru and Liu 2007)	Journal of Systems and Software	Public	Decision Tree	Classification
2008	(Lessmann et al., 2008)	IEEE Transactions on Software Engineering	Public	Random Forest, LR, LDA	Classification
	(Bibi et al., 2008)	Expert Systems with Applications	Private	Regression via Classification	Estimation
	(Gondra, 2008)	Journal of Systems and Software	Public	Support Vector Machine	Classification
	(Vandecruys et al., 2008)	Journal of Systems and Software	Public	Ant Colony Optimization	Classification
	(Elish and Elish 2008)	Journal of Systems and Software	Public	Support Vector Machine	Classification
2009	(Catal and Diri 2009a)	Expert Systems with Applications	Public	Random Forest	Classification
	(Turhan, Kocak, & Bener, 2009)	Expert Systems with Applications	Private	Static Call Graph Based Ranking	Classification
	(Seiffert, Khoshgoftaar, & Van Hulse, 2009)	IEEE Transactions on Systems, Man, and Cybernetics	Public	Boosting	Classification
	(Khoshgoftaar and Gao 2009)	International Conference on Machine Learning and Applications	Public	Undersampling	Classification
	(Catal and Diri 2009b)	Information Sciences	Public	Random Forest and Naïve Bayes	Classification
	(Turhan, Menzies, Bener, & Di Stefano, 2009)	Empirical Software Engineering	Public	k-Nearest Neighbor	Classification
2010	(Menzies et al., 2010)	Automated Software Engineering	Public	WHICH Meta-learning	Classification
	(Zheng, 2010)	Expert Systems with Applications	Public	Neural Network	Classification
	(Liu, Khoshgoftaar, & Seliya, 2010)	IEEE Transactions on Software Engineering	Public	Genetic Programming	Classification
	(H. Wang, Khoshgoftaar, & Napolitano, 2010)	International Conference on Machine Learning and Applications	Public	Naïve Bayes (Ensemble)	Classification
	(Gayatri et al., 2010)	World Congress on Engineering and Computer Science	Public	Decision Tree	Classification
	(Arisholm et al., 2010)	Journal of Systems and Software	Public	Decision Tree	Classification

2011	(Catal, Sevim, & Diri, 2011)	Expert Systems with Applications	Public	Naïve Bayes	Classification
	(Song et al., 2011)	IEEE Transactions on Software Engineering	Public	Naïve Bayes (FS and BE)	Classification
	(Taghi M. Khoshgoftaar, Van Hulse, & Napolitano, 2011)	IEEE Transactions on Systems, Man, and Cybernetics	Public	Bagging	Classification
	(Catal, Alan, & Balkan, 2011)	Information Sciences	Public	Naïve Bayes (LogNum)	Classification
	(R. H. Chang, Mu, & Zhang, 2011)	Journal of Software	Public	Non-Negative Matrix Factorization	Classification
	(Mısırlı et al., 2011)	Software Quality Journal	Public	Naïve Bayes (Ensemble)	Classification
	(Azar and Vybihal 2011)	Information and Software Technology	Private	Ant Colony Optimizzation	Classification
2012	(Gray et al., 2012)	IET Software	Public	-	Dataset Analysis
	(Ying Ma, Luo, Zeng, & Chen, 2012)	Information and Software Technology	Public	Transfer Naïve Bayes	Classification
	(Benaddy and Wakrim 2012)	International Journal of Software Engineering and Its Applications	Private	Neural Network (SA)	Estimation
	(Wong, Debroy, Golden, Xu, & Thuraisingham, 2012)	IEEE Transactions on Reliability	Private	Neural Network (RBF)	Classification
	(Y. Peng, Wang, & Wang, 2012)	Information Sciences	Public	Decision Tree	Classification
	(Zhang and Chang 2012)	International Conference on Natural Computation	Private	Neural Network	Estimation
	(Bishnu and Bhattacherjee 2012)	IEEE Transactions on Knowledge and Data Engineering	Private	k-Means	Clustering
	(Sun, Song, & Zhu, 2012)	IEEE Transactions on Systems, Man, and Cybernetics	Public	Decision Tree	Classification
	(Pelayo and Dick 2012)	IEEE Transactions on Reliability	Public	Undersampling	Classification
	(Jin, Jin, & Ye, 2012)	IET Software	Public	Support Vector Machine	Classification
	(Cao, Qin, & Feng, 2012)	Advanced Science Letters	Public	Neural Network	Classification
2013	(Park et al., 2013)	Information Sciences	Public	Neural Network	Classification
	(Dejaeger, Verbraken, & Baesens, 2013)	IEEE Transactions on Software Engineering	Public	Naïve Bayes	Classification
	(Shepperd, Song, Sun, & Mair, 2013)	IEEE Transactions on Software Engineering	Public	-	Dataset Analysis
	(Wang and Yao 2013)	IEEE Transactions on Reliability	Public	Adaboost	Classification
	(Peters, Menzies, Gong, & Zhang, 2013)	IEEE Transactions on Software Engineering	Public	-	Dataset Analysis
	(Radjenović et al., 2013)	Information and Software Technology	Public	-	Dataset Analysis

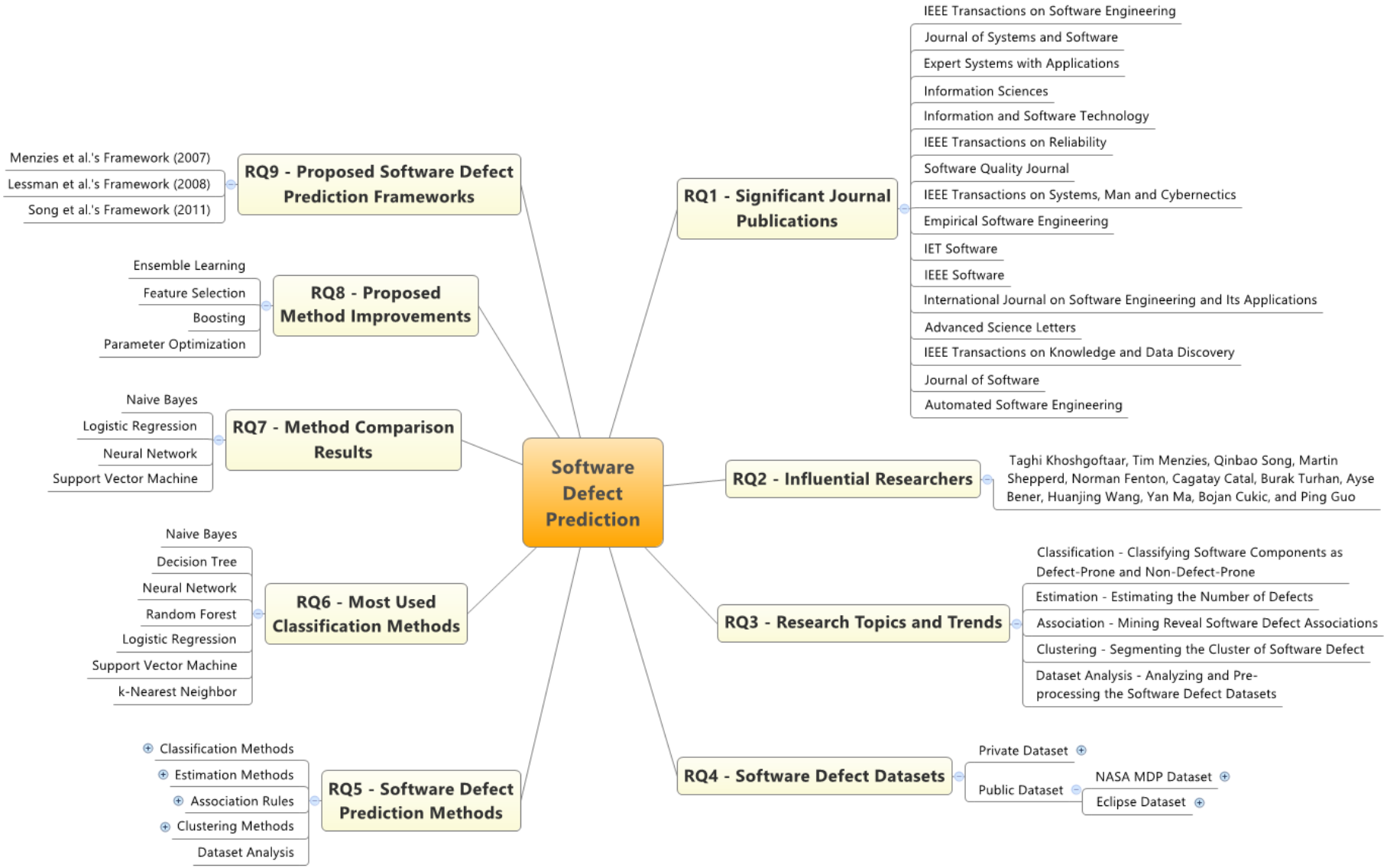


Figure 16 Complete Mind Map of the SLR on Software Defect Prediction

REFERENCES

Arisholm, E., Briand, L. C., & Fuglerud, M. (2007). Data Mining Techniques for Building Fault-proneness Models in Telecom Java Software. *Proceedings of the The 18th IEEE International Symposium on Software Reliability*, 215–224. http://doi.org/10.1109/ISSRE.2007.22

Arisholm, E., Briand, L. C., & Johannessen, E. B. (2010). A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, *83*(1), 2–17. http://doi.org/10.1016/j.jss.2009.06.055

Azar, D., & Vybihal, J. (2011). An ant colony optimization algorithm to improve software quality prediction models: Case of class stability. *Information and Software Technology*, *53*(4), 388–393. http://doi.org/10.1016/j.infsof.2010.11.013

Benaddy, M., & Wakrim, M. (2012). Simulated Annealing Neural Network for Software Failure Prediction. *International Journal of Software Engineering and Its Applications*, *6*(4).

Bibi, S., Tsoumakas, G., Stamelos, I., & Vlahavas, I. (2008). Regression via Classification applied on software defect estimation. *Expert Systems with Applications*, *34*(3), 2091–2101. http://doi.org/10.1016/j.eswa.2007.02.012

Bishnu, P. S., & Bhattacherjee, V. (2012). Software Fault Prediction Using Quad Tree-Based K-Means Clustering Algorithm. *IEEE Transactions on Knowledge and Data Engineering*, *24*(6), 1146–1150. http://doi.org/10.1109/TKDE.2011.163

Boehm, B., & Basili, V. R. (2001). Top 10 list [software development]. *Computer*, *34*(1), 135–137.

Buzan, T., & Griffiths, C. (2013). *Mind Maps for Business: Using the ultimate thinking tool to revolutionise how you work (2nd Edition)*. FT Press.

Cano, J. R., Herrera, F., & Lozano, M. (2003). Using evolutionary algorithms as instance selection for data reduction in KDD: an experimental study. *IEEE Transactions on Evolutionary Computation*, *7*(6), 561–575.

Cao, H., Qin, Z., & Feng, T. (2012). A Novel PCA-BP Fuzzy Neural Network Model for Software Defect Prediction. *Advanced Science Letters*, *9*(1), 423–428.

Catal, C. (2011). Software fault prediction: A literature review and current trends. *Expert Systems with Applications*, *38*(4), 4626–4636.

Catal, C., Alan, O., & Balkan, K. (2011). Class noise detection based on software metrics and ROC curves. *Information Sciences*, *181*(21), 4867–4877.

Catal, C., & Diri, B. (2009a). A systematic review of software fault prediction studies. *Expert Systems with Applications*, *36*(4), 7346–7354.

Catal, C., & Diri, B. (2009b). Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. *Information Sciences*, *179*(8), 1040–1058. http://doi.org/10.1016/j.ins.2008.12.001

Catal, C., Sevim, U., & Diri, B. (2011). Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm. *Expert Systems with Applications*, *38*(3), 2347–2353. http://doi.org/10.1016/j.eswa.2010.08.022

Challagulla, V., Bastani, F., & Yen, I. (2006). A Unified Framework for Defect Data Analysis Using the MBR Technique. *2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, 39–46. http://doi.org/10.1109/ICTAI.2006.23

Challagulla, V. U. B., Bastani, F. B., & Paul, R. A. (2004). Empirical Assessment of Machine Learning based Software Defect Prediction Techniques. In *10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems* (pp. 263–270). IEEE. http://doi.org/10.1109/WORDS.2005.32

Chang, C.-P., Chu, C.-P., & Yeh, Y.-F. (2009). Integrating in-process software defect prediction with association mining to discover defect pattern. *Information and Software Technology*, *51*(2), 375–384. http://doi.org/10.1016/j.infsof.2008.04.008

Chang, R. H., Mu, X. D., & Zhang, L. (2011). Software Defect Prediction Using Non-Negative Matrix Factorization. *Journal of Software*, *6*(11), 2114–2120. http://doi.org/10.4304/jsw.6.11.2114-2120

Cukic, B., & Singh, H. (2004). Robust Prediction of Fault-Proneness by Random Forests. *15th International Symposium on Software Reliability Engineering*, 417–428. http://doi.org/10.1109/ISSRE.2004.35

Dejaeger, K., Verbraken, T., & Baesens, B. (2013). Toward Comprehensible Software Fault Prediction Models Using Bayesian Network Classifiers. *IEEE Transactions on Software Engineering*, *39*(2), 237–257. http://doi.org/10.1109/TSE.2012.20

Denaro, G. (2000). Estimating software fault-proneness for tuning testing activities. In *Proceedings of the 22nd International Conference on Software engineering - ICSE '00* (pp. 704–706). New York, New York, USA: ACM Press.

El Emam, K., & Laitenberger, O. (2001). Evaluating capture-recapture models with two inspectors. *IEEE Transactions on Software Engineering*, *27*(9), 851–864. http://doi.org/10.1109/32.950319

El Emam, K., Melo, W., & Machado, J. C. (2001). The prediction of faulty classes using object-oriented design metrics. *Journal of Systems and Software*, *56*(1), 63–75. http://doi.org/10.1016/S0164-1212(00)00086-8

Elish, K. O., & Elish, M. O. (2008). Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software*, *81*(5), 649–660. http://doi.org/10.1016/j.jss.2007.07.040

Fenton, N. E., & Neil, M. (1999). A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, *25*(5), 675–689. http://doi.org/10.1109/32.815326

Fenton, N., Krause, P., & Neil, M. (2001). A Probabilistic Model for Software Defect Prediction. *IEEE Transactions on Software Engineering*, *44*(0), 1–35.

Fenton, N., Neil, M., Marsh, W., Hearty, P., Marquez, D., Krause, P., & Mishra, R. (2007). Predicting software defects in varying development lifecycles using Bayesian nets. *Information and Software Technology*, *49*(1), 32–43. http://doi.org/10.1016/j.infsof.2006.09.001

Gayatri, N., Reddy, S., & Nickolas, A. V. (2010). Feature Selection Using Decision Tree Induction in Class level Metrics Dataset for Software Defect Predictions. *Lecture Notes in Engineering and Computer Science*, *2186*(1), 124–129.

Gondra, I. (2008). Applying machine learning to software fault-proneness prediction. *Journal of Systems and Software*, *81*(2), 186–195. http://doi.org/10.1016/j.jss.2007.05.035

Gray, D., Bowes, D., Davey, N., & Christianson, B. (2011). The misuse of the NASA Metrics Data Program data sets for automated software defect prediction. *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)*, 96–103.

Gray, D., Bowes, D., Davey, N., Sun, Y., & Christianson, B. (2012). Reflections on the NASA MDP data sets. *IET Software*, *6*(6), 549.

Güneş Koru, a., & Liu, H. (2007). Identifying and characterizing change-prone classes in two large-scale open-source products. *Journal of Systems and Software*, *80*(1), 63–73. http://doi.org/10.1016/j.jss.2006.05.017

Güneş Koru, A., & Tian, J. (2003). An empirical comparison and characterization of high defect and high complexity modules. *Journal of Systems and Software*, *67*(3), 153–163. http://doi.org/10.1016/S0164-1212(02)00126-7

Guo, L., Cukic, B., & Singh, H. (2003). Predicting fault prone modules by the Dempster-Shafer belief networks. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering, 2003* (pp. 249–252). IEEE Comput. Soc. http://doi.org/10.1109/ASE.2003.1240314

Guo, X. C., Yang, J. H., Wu, C. G., Wang, C. Y., & Liang, Y. C. (2008). A novel LS-SVMs hyper-parameter selection based on particle swarm optimization. *Neurocomputing*, *71*(16-18), 3211–3215. http://doi.org/10.1016/j.neucom.2008.04.027

Hall, M. A., & Holmes, G. (2003). Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions on Knowledge and Data Engineering*, *15*(6), 1437–1447.

Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Transactions on Software Engineering*, *38*(6), 1276–1304.

IEEE. (1990). *IEEE Standard Glossary of Software Engineering Terminology* (Vol. 121990). Inst. of Electrical and Electronical Engineers.

J. Pai, G., & Bechta Dugan, J. (2007). Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods. *IEEE Transactions on Software Engineering*, *33*(10), 675–686. http://doi.org/10.1109/TSE.2007.70722

Jiang, Y., Li, M., Zhou, Z., & Member, S. (2011). Software Defect Detection with rocus. *Journal of Computer Science and Technology*, *26*(2), 328–342. http://doi.org/10.1007/s11390-011-1135-6

Jin, C., Jin, S.-W., & Ye, J.-M. (2012). Artificial neural network-based metric selection for software fault-prone prediction model. *IET Software*, *6*(6), 479. http://doi.org/10.1049/iet-sen.2011.0138

Jones, C., & Bonsignour, O. (2012). *The Economics of Software Quality*. Pearson Education, Inc.

Jorgensen, M., & Shepperd, M. (2007). A Systematic Review of Software Development Cost Estimation Studies. *IEEE Transactions on Software Engineering*, *33*(1).

Kanmani, S., Uthariaraj, V. R., Sankaranarayanan, V., & Thambidurai, P. (2004). Object oriented software quality prediction using general regression neural networks. *ACM SIGSOFT Software Engineering Notes*, *29*(5), 1. http://doi.org/10.1145/1022494.1022515

Karthik, R., & Manikandan, N. (2010). Defect association and complexity prediction by mining association and clustering rules. *2010 2nd International Conference on Computer Engineering and Technology*, V7–569–V7–573. http://doi.org/10.1109/ICCET.2010.5485608

Kenny, G. Q. (1993). Estimating defects in commercial software during operational use. *IEEE Transactions on Reliability*, *42*(1), 107–115.

Khoshgoftaar, T. M., & Allen, E. B. (2000). Prediction of software faults using fuzzy nonlinear regression modeling. *Proceedings. Fifth IEEE International Symposium on High Assurance Systems Engineering (HASE 2000)*, 281–290. http://doi.org/10.1109/HASE.2000.895473

Khoshgoftaar, T. M., Allen, E. B., Hudepohl, J. P., & Aud, S. J. (1997). Application of neural networks to software quality modeling of a very large telecommunications system. *IEEE Transactions on Neural Networks / a Publication of the IEEE Neural Networks Council*, *8*(4), 902–9. http://doi.org/10.1109/72.595888

Khoshgoftaar, T. M., Allen, E. B., Jones, W. D., & Hudepohl, J. P. (2000). Classification-tree models of software-quality over multiple releases. *IEEE Transactions on Reliability*, *49*(1), 4–11. http://doi.org/10.1109/24.855532

Khoshgoftaar, T. M., & Gao, K. (2009). Feature Selection with Imbalanced Data for Software Defect Prediction. *2009 International Conference on Machine Learning and Applications*, 235–240. http://doi.org/10.1109/ICMLA.2009.18

Khoshgoftaar, T. M., & Seliya, N. (2002). Tree-based software quality estimation models for fault prediction. *Proceedings Eighth IEEE Symposium on Software Metrics*, 203–214. http://doi.org/10.1109/METRIC.2002.1011339

Khoshgoftaar, T. M., Seliya, N., & Gao, K. (2005). Assessment of a New Three-Group Software Quality Classification Technique: An Empirical Case Study. *Empirical Software Engineering*, *10*(2), 183–218.

Khoshgoftaar, T. M., Seliya, N., & Sundaresh, N. (2006). An empirical study of predicting software faults with case-based reasoning. *Software Quality Journal*, *14*(2), 85–111. http://doi.org/10.1007/s11219-006-7597-z

Khoshgoftaar, T. M., & Van Hulse, J. (2009). Empirical Case Studies in Attribute Noise Detection. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *39*(4), 379–388.

Khoshgoftaar, T. M., Van Hulse, J., & Napolitano, A. (2011). Comparing Boosting and Bagging Techniques With Noisy and Imbalanced Data. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, *41*(3), 552–568.

Kitchenham, B., & Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. *EBSE Technical Report Version 2.3*, *EBSE-2007*-.

Koru, A. G., & Liu, H. (2005). An investigation of the effect of module size on defect prediction using static measures. In *Proceedings of the 2005 workshop on Predictor models in software engineering - PROMISE '05* (Vol. 30, pp. 1–5). New York, New York, USA: ACM Press. http://doi.org/10.1145/1082983.1083172

Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *IEEE Transactions on Software Engineering*, *34*(4), 485–496.

Li, Z., & Reformat, M. (2007). A practical method for the software fault-prediction. In *2007 IEEE International Conference on Information Reuse and Integration* (pp. 659–666). IEEE. http://doi.org/10.1109/IRI.2007.4296695

Lin, S.-W., Ying, K.-C., Chen, S.-C., & Lee, Z.-J. (2008). Particle swarm optimization for parameter determination and feature selection of support vector machines. *Expert Systems with Applications*, *35*(4), 1817–1824. http://doi.org/10.1016/j.eswa.2007.08.088

Liu, Y., Khoshgoftaar, T. M., & Seliya, N. (2010). Evolutionary Optimization of Software Quality Modeling with Multiple Repositories. *IEEE Transactions on Software Engineering*, *36*(6), 852–864.

Lyu, M. R. (2000). Software quality prediction using mixture models with EM algorithm. In *Proceedings First Asia-Pacific Conference on Quality Software* (pp. 69–78). IEEE Comput. Soc. http://doi.org/10.1109/APAQ.2000.883780

Ma, Y., Guo, L., & Cukic, B. (2007). A Statistical Framework for the Prediction of Fault-Proneness. In *Advances in Machine Learning Applications in Software Engineering* (pp. 1–26).

Ma, Y., Luo, G., Zeng, X., & Chen, A. (2012). Transfer learning for cross-company software defect prediction. *Information and Software Technology*, *54*(3), 248–256. http://doi.org/10.1016/j.infsof.2011.09.007

Maimon, O., & Rokach, L. (2010). *Data Mining and Knolwedge Discovery Handbook Second Edition*. Springer.

McDonald, M., Musson, R., & Smith, R. (2007). The practical guide to defect prevention. *Control*, 260–272.

Mende, T., & Koschke, R. (2009). Revisiting the evaluation of defect prediction models. *Proceedings of the 5th International Conference on Predictor Models in Software Engineering - PROMISE '09*, 1. http://doi.org/10.1145/1540438.1540448

Menzies, T., DiStefano, J., Orrego, A. S., & Chapman, R. (2004). Assessing predictors of software defects. In *Proceedings of the Workshop on Predictive Software Models*.

Menzies, T., Greenwald, J., & Frank, A. (2007). Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering*, *33*(1), 2–13.

Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., & Bener, A. (2010). Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engineering*, *17*(4), 375–407.

Mısırlı, A. T., Bener, A. B., & Turhan, B. (2011). An industrial case study of classifier ensembles for locating software defects. *Software Quality Journal*, *19*(3), 515–536. http://doi.org/10.1007/s11219-010-9128-1

Myrtveit, I., Stensrud, E., & Shepperd, M. (2005). Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineering*, *31*(5), 380–391. http://doi.org/10.1109/TSE.2005.58

Naik, K., & Tripathy, P. (2008). *Software Testing and Quality Assurance*. John Wiley & Sons, Inc.

Ostrand, T. J., Weyuker, E. J., & Bell, R. M. (2005). Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, *31*(4), 340–355. http://doi.org/10.1109/TSE.2005.49

Park, B., Oh, S., & Pedrycz, W. (2013). The design of polynomial function-based neural network predictors for detection of software defects. *Information Sciences*, *229*, 40–57.

Pelayo, L., & Dick, S. (2012). Evaluating Stratification Alternatives to Improve Software Defect Prediction. *IEEE Transactions on Reliability*, *61*(2), 516–525. http://doi.org/10.1109/TR.2012.2183912

Peng, J., & Wang, S. (2010). Parameter Selection of Support Vector Machine based on Chaotic Particle Swarm Optimization Algorithm. *Electrical Engineering*, 3271–3274.

Peng, Y., Wang, G., & Wang, H. (2012). User preferences based software defect detection algorithms selection using MCDM. *Information Sciences*, *191*, 3–13. http://doi.org/10.1016/j.ins.2010.04.019

Peters, F., Menzies, T., Gong, L., & Zhang, H. (2013). Balancing Privacy and Utility in Cross-Company Defect Prediction. *IEEE Transactions on Software Engineering*, *39*(8), 1054–1068. http://doi.org/10.1109/TSE.2013.6

Pizzi, N. J., Summers, A. R., & Pedrycz, W. (2002). Software quality prediction using median-adjusted class labels. *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, (1), 2405–2409. http://doi.org/10.1109/IJCNN.2002.1007518

Quah, T., Mie, M., Thwin, T., & Quah, T. (2003). *Application of neural networks for software quality prediction using object-oriented metrics*. *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.* IEEE Comput. Soc.

Radjenović, D., Heričko, M., Torkar, R., & Živkovič, A. (2013, August). Software fault prediction metrics: A systematic literature review. *Information and Software Technology*. http://doi.org/10.1016/j.infsof.2013.02.009

Sammut, C., & Webb, G. I. (2011). *Encyclopedia of Machine Learning*. Springer.

Sandhu, P. S., Kumar, S., & Singh, H. (2007). Intelligence System for Software Maintenance Severity Prediction. *Journal of Computer Science*, *3*(5), 281–288. http://doi.org/10.3844/jcssp.2007.281.288

Seiffert, C., Khoshgoftaar, T. M., & Van Hulse, J. (2009). Improving Software-Quality Predictions With Data Sampling and Boosting. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, *39*(6), 1283–1294.

Seliya, N., & Khoshgoftaar, T. M. (2007). Software Quality Analysis of Unlabeled Program Modules With Semisupervised Clustering. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, *37*(2), 201–211. http://doi.org/10.1109/TSMCA.2006.889473

Shepperd, M., Cartwright, M., & Mair, C. (2006). Software defect association mining and defect correction effort prediction. *IEEE Transactions on Software Engineering*, *32*(2), 69–82. http://doi.org/10.1109/TSE.2006.1599417

Shepperd, M., & Kadoda, G. (2001). Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, *27*(11), 1014–1022. http://doi.org/10.1109/32.965341

Shepperd, M., Song, Q., Sun, Z., & Mair, C. (2013). Data Quality: Some Comments on the NASA Software Defect Datasets. *IEEE Transactions on Software Engineering*, *39*(9), 1208–1215. http://doi.org/10.1109/TSE.2013.11

Song, Q., Jia, Z., Shepperd, M., Ying, S., & Liu, J. (2011). A General Software Defect-Proneness Prediction Framework. *IEEE Transactions on Software Engineering*, *37*(3), 356–370.

Sun, Z., Song, Q., & Zhu, X. (2012). Using Coding-Based Ensemble Learning to Improve Software Defect Prediction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *42*(6), 1806–1817. http://doi.org/10.1109/TSMCC.2012.2226152

Tosun, A., Turhan, B., & Bener, A. (2008). Ensemble of software defect predictors. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '08* (p. 318). New York, New York, USA: ACM Press. http://doi.org/10.1145/1414004.1414066

Turhan, B., Kocak, G., & Bener, A. (2009). Data mining source code for locating software bugs: A case study in telecommunication industry. *Expert Systems with Applications*, *36*(6), 9986–9990. http://doi.org/10.1016/j.eswa.2008.12.028

Turhan, B., Menzies, T., Bener, A. B., & Di Stefano, J. (2009). On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, *14*(5), 540–578. http://doi.org/10.1007/s10664-008-9103-7

Unterkalmsteiner, M., Gorschek, T., Islam, A. K. M. M. K. M. M., Cheng, C. K., Permadi, R. B., & Feldt, R. (2012). Evaluation and Measurement of Software Process Improvement—A Systematic Literature Review. *IEEE Transactions on Software Engineering*, *38*(2), 398–424. http://doi.org/10.1109/TSE.2011.26

Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M., & Haesen, R. (2008). Mining software repositories for comprehensible software fault prediction models. *Journal of Systems and Software*, *81*(5), 823–839. http://doi.org/10.1016/j.jss.2007.07.034

Wang, H., Khoshgoftaar, T. M., & Napolitano, A. (2010). A Comparative Study of Ensemble Feature Selection Techniques for Software Defect Prediction. *2010 Ninth International Conference on Machine Learning and Applications*, 135–140.

Wang, Q., & Yu, B. (2004). Extract rules from software quality prediction model based on neural network. *16th IEEE International Conference on Tools with Artificial Intelligence*, (Ictai), 191–195. http://doi.org/10.1109/ICTAI.2004.62

Wang, S., & Yao, X. (2013). Using Class Imbalance Learning for Software Defect Prediction. *IEEE Transactions on Reliability*, *62*(2), 434–443.

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data Mining Third Edition*. Elsevier Inc.

Wong, W. E., Debroy, V., Golden, R., Xu, X., & Thuraisingham, B. (2012). Effective Software Fault Localization Using an RBF Neural Network. *IEEE Transactions on Reliability*, *61*(1), 149–169. http://doi.org/10.1109/TR.2011.2172031

Xing, F., Guo, P., & Lyu, M. R. (2005). A Novel Method for Early Software Quality Prediction Based on Support Vector Machine. *16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05)*, 213–222. http://doi.org/10.1109/ISSRE.2005.6

Zhang, P., & Chang, Y. (2012). Software fault prediction based on grey neural network. In *2012 8th International Conference on Natural Computation* (pp. 466–469). IEEE. http://doi.org/10.1109/ICNC.2012.6234505

Zheng, J. (2010). Cost-sensitive boosting neural networks for software defect prediction. *Expert Systems with Applications*, *37*(6), 4537–4543.

Zhou, Y., & Leung, H. (2006). Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. *IEEE Transactions on Software Engineering*, *32*(10), 771–789. http://doi.org/10.1109/TSE.2006.102

BIOGRAPHY OF AUTHOR

Romi Satria Wahono. Received B.Eng and M.Eng degrees in Computer Science respectively from Saitama University, Japan, and Ph.D in Software Engineering and Machine Learning from Universiti Teknikal Malaysia Melaka. He is a lecturer at the Faculty of Computer Science, Dian Nuswantoro University, Indonesia. He is also a founder and chief executive officer of PT Brainmatics Cipta Informatika, a software development company in Indonesia. His current research interests include software engineering and machine learning. Professional member of the ACM, PMI and IEEE Computer Society.

A Systematic Literature Review of Requirements Engineering for Self-Adaptive Systems

Slamet Sucipto
Department of Informatics Engineering, STMIK Eresha
takwa.inspiration@gmail.com

Romi Satria Wahono
Faculty of Computer Science, Dian Nuswantoro University
romi@romisatriawahono.net

Abstract: During 2003 to 2013, the continuous effort of researchers and engineers particularly has resulted in a hugely grown body of work on engineering self-adaptive systems. Although existing studies have explored various aspects of this topic, no systematic study has been performed on categorizing and evaluating the requirement engineering for self-adaptive activities. The objective of this paper is to systematically investigate the research literature of requirements engineering for self-adaptive systems, summarize the research trends, categorize the used modeling methods and requirements engineering activities as well as the topics that most described. a systematic literature review has been conducted to answer the research questions by searching relevant studies, appraising the quality of these studies and extracting available data. From the study, a number of recommendations for future research in requirements engineering for self-adaptive systems has been derived. So that, enabling researchers and practitioners to better understand the research trends.

Keywords: self-adaptive, requirement engineering, systematic literature review

1 INTRODUCTION

In the last decade, there are the increasing cost of handling the complexity of software systems to achieve robustness in handling unexpected conditions, changing priorities and policies governing the goals, and changing conditions (Laddaga, 2001). Welsh and Sawyer (2010) argued that the uncertainty about the environments is the reason why a system must be able to continue to operate in a range of contexts with different requirements or requirements trade-offs (Welsh & Sawyer, 2010). Cope with the increasing complexity, software systems must become more versatile, flexible, resilient, dependable, energy-efficient, recoverable, customizable, configurable and self-optimizing by adapting to changes that may occur in their operational contexts, environments and system requirements (Lemos, Giese, & Müller, 2013). From the explanation above, noted that the reason why the self adaptive systems should be conducted, are include: the cost handling of the complexity of systems to cope with unexpected condition and change goal priorities, the uncertainty about the environments that need the different requirements or requirements trade-offs, the increasing complexity of system inflict the system requirements more flexible, effective and efficient.

The self-adaptive is the capability of the system to adjust its behavior in response to the environment which the systems autonomously decide how to adapt or to organize themselves so that they can accommodate changes in their contexts and environments (Brun, Serugendo, & Gacek, 2009). Laddaga in (Laddaga, 2001) argued that self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible. The conception of self-adaptive system depends on user's requirements, system properties and environmental characteristics.

The primary appraise of success of a software system is the degree to which it meets the purpose for which it was intended (Kaur & Singh, 2010). Understandings the requirements of an adaptive software system is critical to successful development and deployment, as a means of taking advantage of adaptation semantics that describe how systems behave during adaptation (Macías-Escrivá, Haber, del Toro, & Hernandez, 2013). Deal with the uncertainty of unanticipated contexts that prompt new requirements, Sawyer et al. (2010) state that requirements for self-adaptive systems should be run-time entities that can be reasoned to understand requirements are satisfied, and to support adaptation decisions that can take advantage of the system (Sawyer, Bencomo, Whittle, Letier, & Finkelstein, 2010). Although software engineering still lacks a mature science of software behavior on which to draw, requirements engineers need such a science in order to understand how to specify the required behavior of software (Kaur & Singh, 2010). The important points of development and deployment of self-adaptive system successful, are included: understandings the requirements of an adaptive software system, the requirements for self-adaptive systems should be run-time entities, the needs a science in order to understand how to specify the required behavior of software.

Requirements engineering (RE) is known as the first stage in the lifecycle of software development, aiming at defining domain logic, identifying stakeholders' needs and documenting information for subsequent analysis and implementation (Nuseibeh & Easterbrook, 2000). Lamsweerde (2008) argued that requirements engineering is concerned with the elicitation, evaluation, specification, consolidation, and evolution of the objectives, functionalities, qualities, and constraints a software based system should meet within some organizational or physical setting (Van Lamsweerde, 2008).

Different from traditional requirement engineering, in (Yang, Li, Jin, & Chen, 2014) state that requirement engineering for self-adaptive systems focuses more on defining adaptation logic, since the self-adaptive systems need adaptation mechanisms. In consequence, the description of what to adapt, when to adapt, what changes in the environment and the system to be monitored and how to adapt during

requirement engineering for self-adaptive must be addressed by engineers.

During 2003 to 2013, the continuous effort of researchers and engineers particularly has resulted in a hugely grown body of work on engineering selfadaptive systems. In (Ganek & Corbi, 2003) has presented an overview of IBM's autonomic computing initiative. It examines the genesis of autonomic computing, the industry and marketplace drivers, the fundamental characteristics of autonomic systems, a framework for how systems will evolve to become more self-managing, and the key role for open industry standards needed to support autonomic behavior in heterogeneous system environments. Dobson et al. (2009) surveyed the state of autonomic communications research and identify significant emerging trends and techniques. As a general conclusion, these techniques fundamentally change the ways in which those designing, implementing, deploying, administering, and using highly-distributed adaptive systems will interact with those systems in the future (Dobson et al., 2006). Brun et al. (2009) explore the state-of-the-art in engineering self-adaptive systems and identify potential improvements in the design process. Researchers in this study argue that feedback loops are a key factor in software engineering of self-adaptive systems (Brun et al., 2009). Cheng et al. (2009) present the summary of state-of-the-art and to identify critical challenges for the systematic software engineering of self-adaptive systems. In this study researchers argue that the engineering of self-adaptive software systems is a major challenge and feedback loops is a major property in self-adaptive systems (Cheng, Lemos, & Giese, 2009). Salehie and Tahvildari (2009) have presented a landscape of research in self-adaptive software by highlighting relevant disciplines and some prominent research projects, as well as helps to identify the underlying research gaps and elaborates on the corresponding challenges. The research presents a taxonomy, based on concerns of adaptation, that is, how, what, when and where, towards providing a unified view of this emerging area (Salehie & Tahvildari, 2009). lemos et al. (2013) present the summary of state-of-the-art and identify research challenges when developing, deploying and managing self-adaptive software systems, as well as complements and extends a previous roadmap on software engineering for self-adaptive systems published in (Cheng et al., 2009). In these study, researcher argue that all these challenges result from the dynamic nature of self-adaptation, which brings uncertainty to the forefront of system design. (Lemos et al., 2013). Macías-Escrivá et al (2013) explore the review of recent progress on self-adaptivity based on the analysis of state-of-the-art approaches reported in the literature. The review provides an overarching integrated view of computer science and software engineering foundations. Moreover, various methods and techniques currently applied in the design of self-adaptive systems are analyzed, as well as some European research initiatives and projects. Finally, the main bottlenecks for the effective application of self-adaptive technology, as well as a set of key research issues on this topic, are precisely identified, in order to overcome current constraints on the effective application of self-adaptivity in its emerging areas of application (Macías-Escrivá et al., 2013).

However, no systematic study has been performed on categorizing and evaluating the requirement engineering for self-adaptive activities. Thus, there is no clear view on where the researches are conducted and where the results are published, what extend requirement engineering for self-adaptive activity is studied, how the quality of studies varies against each activity and what the most active topics are.

The objective of this paper is to systematically investigate the research literature of requirements engineering for self-adaptive systems, summarize the research trends, categorize the used modeling methods and requirements engineering activities as well as the topics that most described. To conduct the investigation and report analysis results, the research methodology of systematic literature review in (Kitchenham & Charters, 2007) and (Brereton, Kitchenham, Budgen, Turner, & Khalil, 2007) have been performed.

The rest of the paper is structured as follows. Section 2 briefly describes the research method. Section 3 discusses the results and discussion and threats to validity in Section 4. Related roadmap and survey presented in Section 5, followed by the conclusion and future works in Section 6.

2 RESEARCH METHOD

A systematic literature review (SLR) in (B. A. Kitchenham & Charters, 2007) is a well-defined approach to identify, evaluate and interpret all relevant studies regarding a particular research question, topic area or phenomenon of interest. By following the SLR process, the review protocol divided in seven stages: specify research questions, define search scope and strategy, specify data items, select primary studies, extract required data, analyze data and write review report. Figure 1 shows an overview of the main phases that adopted from (Brereton, Kitchenham, Budgen, Turner, & Khalil, 2007).

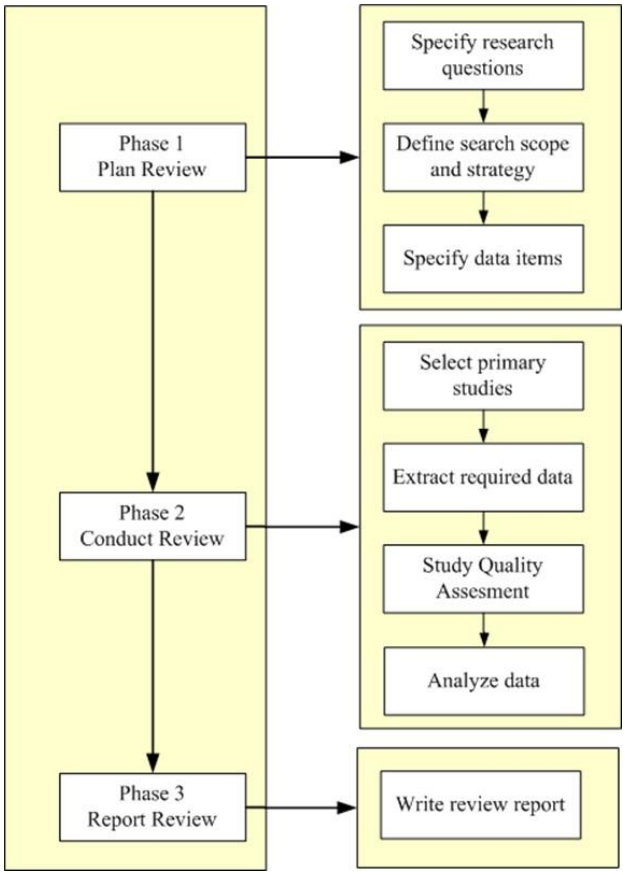


Figure 1. Overview of The Systematic Literature Review

In review planning, the researchers defined the review protocol as described in this document. The review protocol includes the definition of research questions, the search strategy and scope, the data items that had to be collected, and the approach for data analysis and presentation of the results. In the next phase, the researchers have to conduct the review

by through selecting primary studies based on the search criteria and data has to be collected as specified in the protocol defined in phase 1. Furthermore, the data derived from the primary studies has to be collated and summarized to answer the research questions defined in the protocol. Finally, the review report has to be produced in phase 3. The final report will be cross checked by a supervisor researcher that will be used to improve the description and correct minor issues.

2.1 SPECIFY RESEARCH QUESTIONS

To formulate the research questions, the implementation of Goal-Question-Metric (GQM) approach in (Basili, Caldiera, & Rombach, 1994) that provide approach aim at developing meaningful metrics for questions have been conducted. The goal of this literature research is to review the existing research work in the literature of requirements engineering for self-adaptive systems. To achieve this goal by decomposing into four sub-goals:

1. To provide the basic publication information and to assist identifying the most appropriate sources of research information in requirements engineering for self-adaptive systems.
2. To assist identifying the range of methods currently available in requirements engineering for self-adaptive systems.
3. To identify which methods and activities are more widely studied and to provide the hint whether any methods and activities are mature enough to be applied futher.
4. To identify principal research trends and to highlight active research topics that related with requirements engineering for self-adaptive systems.

To achieve the sub-goals, we design 10 research questions (RQ) in Table 1.

Table 1. The Research Questions

Research Quetions		Type	Related Goal
RQ1	What is the time of the publications?	Publication	1
RQ2	What is the veneu of the publications?		
RQ3	Who are the most active and influential researchers?		
RQ4	What activities are most studied?	Content	2
RQ5	What modeling are used to model requirement engineering for self adaptive system?		
RQ6	Which quality attributes can be concerned in Requirement Engineering for Self Adaptive?		
RQ7	What application domain are used for illustration?		
RQ8	Which methods are better applied and have more rigorous evaluation?	Quality	3
RQ9	Which RE activities are presented and discussed more detailedly?		
RQ10	What is the relationship between topics, activities and modeling methods?	Topic	4

To achieve first sub-goal, RQ1, RQ2 and RQ3 supported by providing the basic information related with trend of literature and relevant vanue of publication as well as the most active researchers who contributed on a research area. From RQ4 to RQ7 provide the summaries of analysis activities, modeling, quality attribute and application domain to achieve the second sub-goal. RQ8 and RQ9 achieve the third sub-goal by providing the measurement results of quality studied and activities are discussed detailedly. To fullfill the fourth sub-goal, RQ10 support us by providing the summaries of relationship between topics, activities and modeling method. For the Metric part of the GQM method, we will illustrate how to derive available information in the data extraction section.

2.2 DEFINE SEARCH SCOPE AND STRATEGY

The activities related with search strategy include, selecting digital library, defining the search string, executing a pilot search, refining the search string and retrieving an initial list of primary studies from digital libraries matching the search string. Selecting digital library performed by chosing an appropriate set of databases so that the probability of finding highly relevant papers fullfilled. To ensure thorough retrieval, we choose four popular databases library that cover the requirements engineering for self-adaptive literature, as follows:

1. IEEE eXplore (ieeexplore.ieee.org)
2. ACM Digital Library (dl.acm.org)
3. SpringerLink (springerlink.com)
4. ScienceDirect (sciencedirect.com)

Kitchenham et al. have described the steps for constructing the search string. We present Kitchenham's steps in Figure 2.

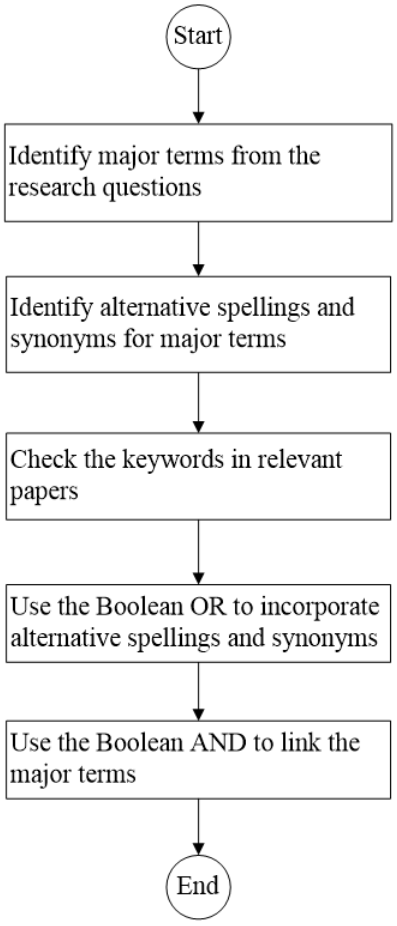


Figure 2. The Construction Steps of Search Strings

The derived search strings presented in Table 2. The use of the search strings can be combined with Boolean operator as: S1 AND S2.

Table 2. Derived Search Strings

S1	S2
Self-Adaptive	Requirements
("self-adaptive systems" OR "dynamically adaptive systems" OR "adaptive system" OR "Adaptive software" OR"self-adaptive software" OR "adaptive service" OR "self-adaptation" OR "socio-technical system" OR "self-adjusting systems" OR "autonomic computing" OR "self-adapting software" OR "self-reconfiguration" OR "self-repair" OR "self-healing" OR "self-tuning" OR "context-awareness")	("model requirements" OR "modeling requirements" OR "Requirements modeling" OR "specify requirements" OR "specifying requirements" OR "requirements specifying" OR "requirements specification" OR "monitor requirements" OR "monitoring requirements" OR "requirements monitoring" OR "aware requirements" OR "requirements-aware" OR "requirements awareness" OR "requirements-awareness" OR "diagnose requirements" OR "diagnosing requirements" OR "requirements diagnosing" OR "requirements diagnosis" OR "detect requirements" OR "detecting requirements" OR"requirements detection" OR "verify requirements" OR "verifying requirements" OR "requirements verifying" OR "requirements verification" OR "satisfy requirements" OR"satisfying requirements" OR "requirements satisfying" OR "requirements satisfaction" OR "evolution requirements" OR "requirements evolution")

The adjustment of the search string are conducted, but the original one is kept, since the adjustment of the search string would dramatically increase the already extensive list of irrelevant studies. The search string was subsequently adjusted to suit the specific requirements of each database. The databases are searched by title, keyword and abstract. The search was limited by the year of publication: 2000-2014. Two kind of publications, journal papers and conference proceedings, were included. The search was limited only to English.

2.3 SPECIFY DATA ITEMS

The data items aim to support availability data in order to answer each research questions. For each primary study, the data items shown in Table 3 have to be fullfilled as long as conduct selecting primary studies.

Table 3. Data Items

ItemId	Field	Use
F1	Title	Documentation
F2	Year	RQ1
F3	Veneu	RQ2
F4	Indexed Scopus	RQ2
F5	Authors	RQ3
F6	Activity	RQ4,RQ10
F7	Modeling Method	RQ5,RQ10
F8	Quality Attribute	RQ6
F9	Application Domain	RQ7
F10	Context	RQ10
F11	Topic	Documentation
F12	High-Level Problem Statement	Documentation
F13	Quality Score of Problem Statement	RQ8,RQ9
F14	Quality Score of Context	RQ8,RQ9
F15	Quality Score of Modeling Method	RQ8,RQ9
F16	Quality Score of Activity	RQ8,RQ9
F17	Quality Score of Application Domain	RQ8,RQ9

According to Table 3, the data items such as title (F1) and high-level problem statement (F11) are used for documentation. To answer RQ1 to RQ3, the data items that used, include: year (F2), venue (F3), indexed scopus (F4) and authors (F5). Activity (F6), modeling method (F7), quality attribute (F8), application domain (F9) are used to answer RQ4 to RQ7. To answer the questions related with quality score such as RQ8 and RQ9, the quality score of data items that used, include: activity (F6), modeling method (F7), application domain (F9), context (F10), topic (F11), and problem staement (F12) jointly.

2.4 SELECT PRIMARY STUDY

The Inclusion criteria and exclusion criteria are defined for selecting relevant studies. Retrieved papers are firstly checked with exclusion criteria.The paper will be excluded if such paper meets any one of the exclusion criteria. The remaining papers are checked with inclusion criteria. If one paper meets all the inclusion criteria then it will be included. These criteria are shown in Table 4.

Table 4. Inclusion and Exclusion Criteria

Inclusion Criteria	Exclusion Criteria
Published time between 2000 - 2014	Publised In form Books, web sites, technical reports, and master theses or short paper (less than 6 pages)
Focus on Requirement Engineering for Self-Adaptive	Papers that Focus on summarizes the existing research work, roadmap or survey
Related with Requirement Engineering Activity	Publications in which focus on RE but is not clearly understandable or covered in insufficient detail
Involve concrete Measurement methods and evaluation to the methods	Studies without a strong validation or including experimental results
Publised in Form Doctoral Dissertation	Publised in Journal that not indexed in Scopus

As shown in Figure 3, the study selection process was conducted in two steps: the exclusion of primary studies based on the title and abstract and the exclusion of primary studies based on the full text. The literature review studies should be excluded when not explain the activity of requirement engineering for self-adaptive. And the other hand, the studies that deal with requirement engineering for self-adaptive as clearly, it must included.

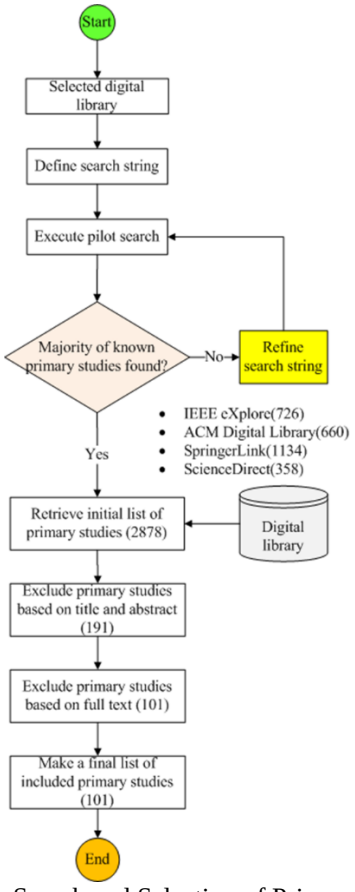


Figure 3. Search and Selection of Primary Studies

By conducting each phase of the search process, the final list of selected primary studies included 101 primary studies.

Then full texts were analyzed for 101 primary studies. In addition to the inclusion and exclusion criteria, the quality of the primary studies, their relevance to the research questions and study similarity were considered. Similar studies by the same authors in various journals were removed. 101 primary studies remained after the exclusion of studies based on the fulltext. The complete list of selected studies is provided in the Section Systematic Literature Review References.

2.5 EXTRACT REQUIRED DATA

Data extraction performed in order to accumulate the corresponding information from the selected primary studies that have been extrated. The corresponding information are collected to the data extraction form that was designed to answer the research questions. For each of the 101 selected primary studies, the data extraction form was completed. In table 5 presented the 4 properties that identified to answer the research questions. The process of data extraction is performed in an iterative manner.

Table 5. Data Extraction Properties Mapped to Research Questions

Property	Research Questions
Publication	RQ1, RQ2, RQ3
Content	RQ4, RQ5, RQ6, RQ7
Quality	RQ8, RQ9
Topic	RQ10

2.6 STUDY QUALITY ASSESMENT

Table 6 defined the quality assessment checklist based on the assessment items that adopted from (B. Kitchenham & Charters, 2007) and (Dybå & Dingsøyr, 2008). The quality of each primary studies is assessed accordance with optional answer score, and then the assessment result is used to answer the quality assement question such as RQ8 dan RQ9. The checklist is not used to evaluate whether a content of study is good or not, but use it to evaluate whether the content of study is maturely presented in the literature.

Table 6. Quality Assessment Checklist

No	Quality Assesment Questions	Optional Answer Score
1	How clearly is the problem of study described?	Explicitly=1; Vague=0.5; No description=0
2	How clearly is the research context stated?	Explicitly=1; Generally=0.67;Vague=0.33; No description=0
3	How detailedly is the modeling method presented?	Explicitly=1; Generally=0.67;Vague=0.33; No description=0
4	How detailedly is the activity of RE in self-adaptive elaborated?	Explicitly=1; Generally=0.67;Vague=0.33; No description=0
5	How clearly is the application domain presented?	Explicitly=1; Generally=0.67;Vague=0.33; No description=0

2.7 ANALYZE DATA

Data Analysis performed by assesing the quality of studies that used to guide the interpretation of the synthesis findings as well as to define the strength of the elaborated inferences. The data analysis aim at aggregating data from the selected studies in order to answer the research questions. The data extracted in this review include both quantitative data and qualitative data. Different strategies are employed to synthesize the extracted data pertaining to different kinds of research questions. The data were tabulated in a manner consistent with the questions. And to enhance the presentation of the distribution of requirement engineering in self-adaptive systems, usage of some visualization tools include bar chart, pie chart, and tables can be utilized. Based on the synthesis, conclusions and recommendations for future research in the field will be derived, and limitations of the review have to be identified. Finally, the results of the review are presented in a review report.

3 THREATS TO VALIDITY

The concentration of this study is to analyze the studies on requirement engineering in self-adaptive systems. This study are not aware of biases may have had for choosing the studies. The paper searching is not based on manual reading of titles of all published papers in journals. It means that this review probably have excluded some related research papers which exist in some conference proceedings or journals.

This study include the papers from conference proceedings, because experience reports are mostly published in conference proceedings. Some systematic literature reviews related with self-adaptive systems, as examples in (Yang et al., 2014), (Weyns, Iftikhar, Malek, & Andersson, 2012) and (Weyns & Ahmad, 2013) that included papers in conference proceedings as the primary studies.

4 RESULT AND DISCUSSION

This Section presents the results of research questions and discuss them briefly one by one. For each research question, we give a graphical overview and also we analyses of our data by statistically. At the end of each research question, we give a summary of that research question, in which we discuss that what we learned from this research question.

4.1 TIME DISTRIBUTION OF THE PUBLICATIONS

The time distribution of the publication is presented to show how the attention in the topic has changed over time. Figure 4 presented the overview of the distribution studies over years.

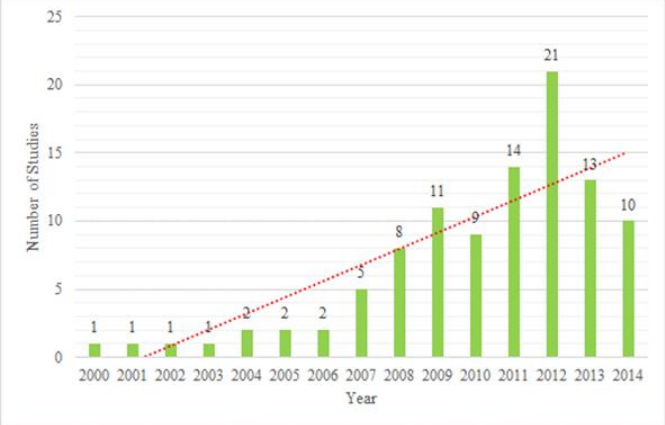


Figure 4. Time Distribution of Publications

Since 2007, an increase of publication caused by increased activity of requirement specification. This increase was driven by improvement in the model approach , as example in (Bresciani, Perini, Giorgini, Giunchiglia, & Mylopoulos, 2004) presented goal-based model that called TROPOS, in (Garlan, Cheng, Huang, Schmerl, & Steenkiste, 2004) presented architecture-based self-adaptation that called RAINBOW that provide model approach in order to fullfill the requirement of

stakeholders and requirements engineering language namely Relax in (Whittle, Sawyer, Bencomo, Cheng, & Bruel, 2010). Figure 4 also shows that the research field on Requirement Engineering in self-adaptive systems is still very much relevant to this day.

4.2 VENEU DISTRIBUTION OF PUBLICATIONS

From 101 primary studies that collected, there are 60 studies from journals, while 41 papers from conference proceedings. Figure 5 and 6 show the percentage of the distribution of studies per venue. Presented that Lecture Notes in Computer Science (LNCS) is the most popular journal to publish papers on Requirement engineering in self adaptive systems with 43% of the studies, while SEAMS is the most prominent conference with 22% of the studies.

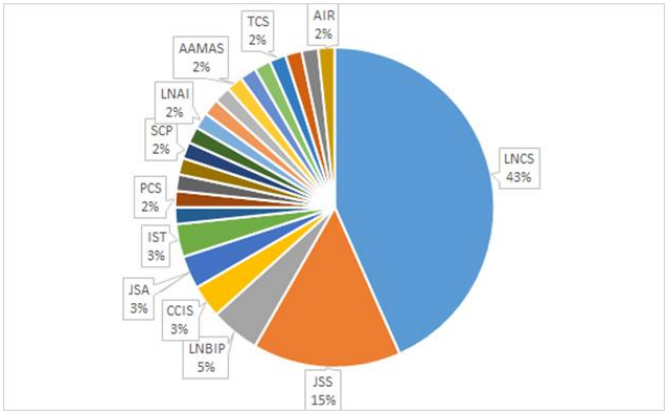


Figure 5. Venue Distribution of Journal

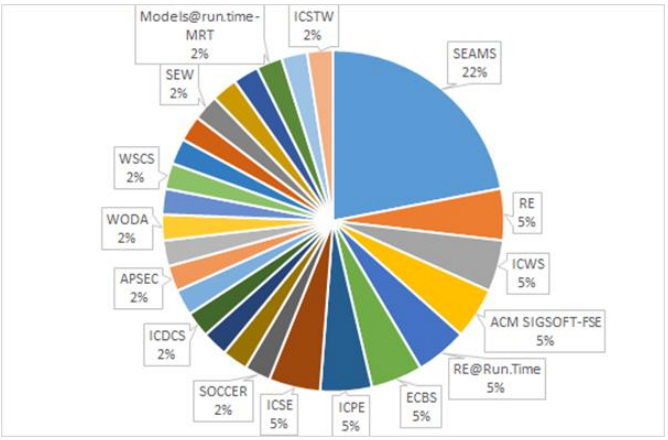


Figure 6. Venue Distribution of Conference Proceedings

Table 7 shows the Scimago Journal Rank (SJR) value and Q categories of the most important journals. Journal publications are ordered by frequency value. Table 8 presents the Scimago Journal Rank and ERA Rank of conference proceedings.

Table 7. Scimago Journal Rank (SJR) of Journal

Journal Article	Freq.	%	HI	SJR	Q Category
LNCS	26	43%	118	0.31	Q2 In Computer Science
JSS	9	15%	60	0.82	Q1 In Hardware and Architecture
LNBIP	3	5%	13	0.25	Q2 In Business, Management and Accounting
IST	2	3%	54	1.07	Q1 In Software
JSA	2	3%	30	0.39	Q3 In Software
CCIS	2	3%	12	0.14	Q4 In Computer Science
AIR	1	2%	40	1.24	Q2 In Artificial Intelligence
AAMAS	1	2%	46	1.57	Q1 In Artificial Intelligence
CACM	1	2%	131	1.82	Q1 In Computer Science
Computer	1	2%	112	0.72	Q1 In Computer Science
CSRD	1	2%	13	0.36	Q2 In Computer Science
ESE	1	2%	39	1.29	Q1 In Software
ESA	1	2%	85	149	Q1 In Computer Science Applications
FAC	1	2%	27	1.05	Q1 In Software
IEEE Software	1	2%	72	0.84	Q3 In Software
JISA	1	2%	7	0.88	Q1 In Computer Networks and Communications
LNAI	1	2%	118	0.31	Q2 In Computer Science
PCS	1	2%	15	0.28	Computer Science (miscellaneous)
RE	1	2%	32	0.98	Q3 In Software
SCP	1	2%	44	0.67	Q2 In Software
TCS	1	2%	74	0.93	Q1 In Computer Science
WIAS	1	2%	15	0.32	Q3 In Computer Networks and Communications
Total	**60**				

Table 8. Scimago Journal Rank (SJR) and ERA Rank of Proceedings

Conference Proceedings	Freq.	%	HI	SJR	ERA Rank
SEAMS	9	22%	7	0	A
ACM SIGSOFT-FSE	2	5%	39	0.74	A
ECBS	2	5%	3	0.17	B
RE	2	5%	4	0.29	A
ICPE	2	5%	2	0.18	N/A
ICWS	2	5%	8	0.2	A
RE@Run.Time	2	5%	2	0.2	N/A
ICSE	2	5%	2	0.1	C
ACM SAC	1	2%	35	0.31	B
APSEC	1	2%	16	0.2	C
Internetware	1	2%	0	0.1	N/A
CIT	1	2%	6	0	C
COMPSAC	1	2%	17	0.19	B
DEAS - ACM SIGSOFT	1	2%	17	0	N/A
EASe	1	2%	2	0.16	A
ACM SIGSOFT-QoSA	1	2%	2	0.17	A
ICDCS	1	2%	3	0.16	A
ICSTW	1	2%	7	0.25	C
Models@run.time - MRT	1	2%	1	0.11	N/A
NOTERE	1	2%	4	0.12	N/A
RAISE	1	2%	2	0.11	N/A
REV	1	2%	1	0	B
SOCCER	1	2%	1	0	C
SEW	1	2%	1	0.1	C
WODA	1	2%	1	0.1	N/A
WSCS	1	2%	4	0	N/A
Total	**41**				

4.3 MOST ACTIVE AND INFLUENTIAL RESEARCHERS

From the selected primary studies, the researchers who contributed very well and who are active on the requirement engineering in self-adaptive systems research field are investigated and identified. Figure 7 shows the active and influential researchers in requirement engineering in self-adaptive systems field. The researchers are listed according to the number of studies included in primary studies. However, it can be noted that Betty Cheng, Pete Sawyer, Anna Perini, John Mylopoulos, Andres Ramirez and Nelly Bencomo are active researchers on requirement engineering in self-adaptive systems.

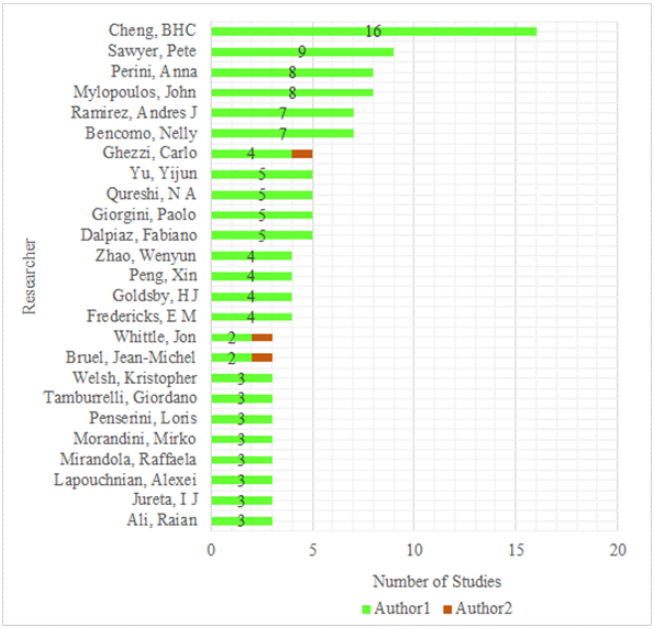


Figure 7. Influential Researchers and Number of Studies

4.4 THE MOST STUDIED ACTIVITIES

Figure 8 presents the categories of RE activities and the corresponding frequency of studies. From the selected primary studies, requirement specification is the activities that most studied on requirement engineering in self-adaptive systems. it can be noted that the 5 activities of research on requirement engineering in self-adaptive systems that most studied, include: requirement specification, requirement modeling, requirement verification, requirement monitoring and adaptation mechanism.

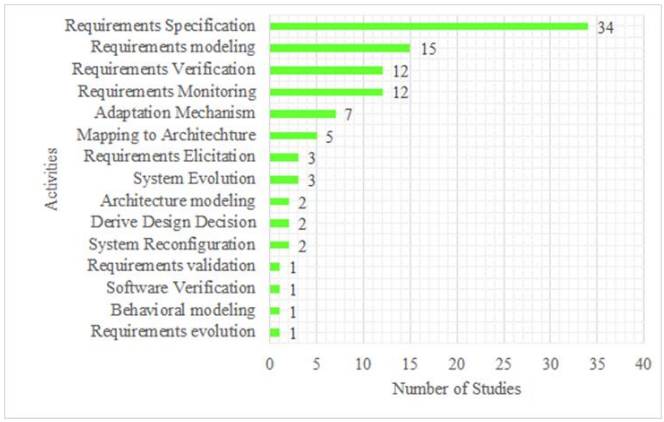


Figure 8. Requirement Engineering Activities based on Frequency of Studies

4.5 THE MOST USED MODELING METHODS

Figure 9 presents the modeling methods and the corresponding frequency of studies. Tropos, i* and KAOS are Goal-oriented methodologies that became most popular requirements modeling methods in the literature. They can describe intension of stakeholders and systems' requirements as clearly. In addition, UML models are used to model behavior of systems. Transition systems including Markov Chain, DTMC, Petri Net and State and Transition System are implemented to describe systems' states and state transitions. Feedback control mechanism are to design the adaptation mechanism. LTL, FBTL and Mathematical Logic are used as specification languages which are utilized to specify the properties that should be held by the system. And to capture the environmental properties can be built by context models. Business process model and domain-specific model focus more on business logic and domain logic.

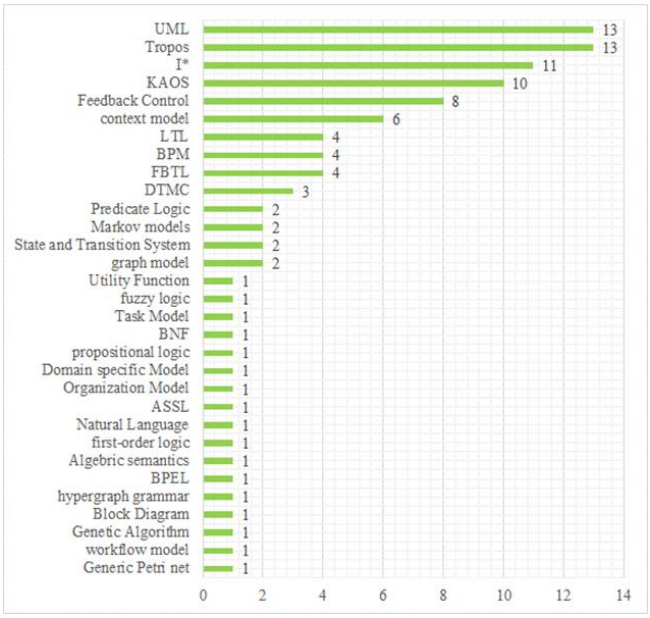


Figure 9. Modeling Methods based on Frequency of Studies

4.6 THE MOST CONCERNED OF QUALITY ATTRIBUTE

In this chapter, quality attributes are investigated based on ISO 9126-1 (http://www.iso.org). Related with definition of each quality attributes do not elaborated, but explain the relations implied behind. Figure 10 presents the distrbution of quality attributes.

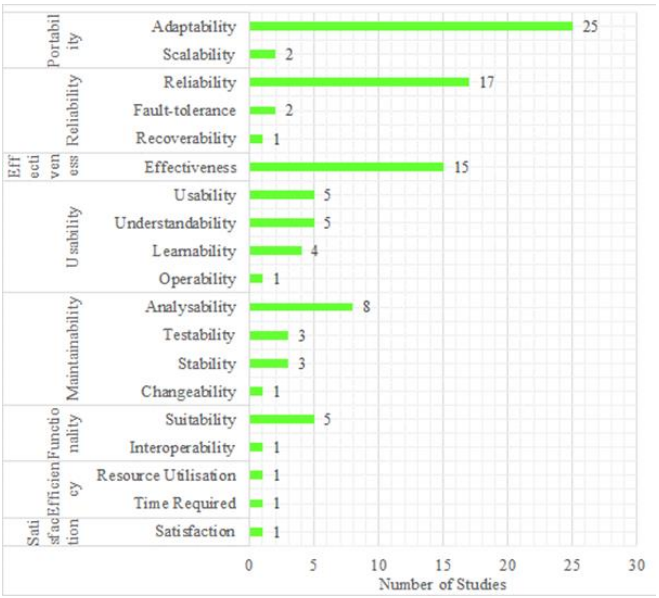


Figure 10. Quality Attributes and Number of Studies

From selected the primary studies, noted that the 7 most concerned of quality attributes on requirement engineering in self adaptive, include: adaptability, reliability, effectiveness, analysability, understandability, usability and suitability. Adaptability is involved in the studies of building adaptation mechanism or runtime adaptation. Reliability is studied in the work on the topic of verification. Effectiveness involved in the studies that specify goals with accuracy and completeness.

Analyzability is considered in the studies of monitoring or diagnosing requirements. Understandability is involved in the study of producing more understandable requirements model. Usability is considered in the studies deal with context-awareness system. Suitability is studied in the work on the topic of formal method that provide an appropriate set of functions for specified tasks and user objectives.

4.7 THE MOST USED OF APPLICATION DOMAIN

From the selected primary studies, noted that 43 application domains have been used to support the experiment to the research of requirement engineering in self-adaptive systems. Figure 11 presents the 15 application domains that most used to illustrate the experiment of research.

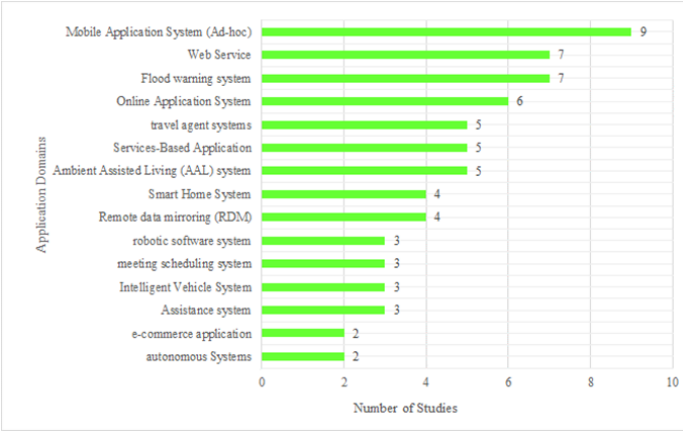


Figure 11. Application Domains and Number of Studies

Mobile application system is application domain that used to the research which deal with specifying requirement on context-awareness system. In the context of service-oriented systems, web service is application domain that illustrate the experiment of the research. In order to support the activities of specification, monitoring and modeling requirements in dynamically adaptive systems (DAS), the researcher most used flood warning system as application domains. Related with the activities that cope with non-functional requirement satisfaction, most researcher used remote data mirroring (RDM) as application domains.

In order to support the research related with non-functional requirement issue, Table 9 describe that remote data mirroring (RDM) is the application domain that most clearly elaborated to support non-functional requirement satisfaction research.

Table 9. Application Domain and Quality Score based on Research Topic

Application Domain	Deal with Non-Functional Requirement	Deal With Trade Off Analysis
Assistance system		4,50
e-commerce application	4,34	
Happy Hour Organizer (HHO)	3,84	
Intelligent Vehicle System	4,00	
load balancer system	4,00	
meeting scheduling system		3,67
Mobile Application System (Ad-hoc)		4,09
Online Application System	3,67	
Personal Emergency Response System (MPERS)	4,17	
Positioning System	4,33	
Remote data mirroring (RDM)	5,00	
Services-Based Application	3,33	
tour guide system	4,34	
urban transportation management	3,67	
Web Service	3,50	3,67

4.8 RIGOROUS EVALUATION METHODS

The primary studies are assessed according to the quality assessment checklist in Table 6. Figure 12 depicts that KAOS, i*, Tropos have high score, because most research topic proposed have clear evaluation.

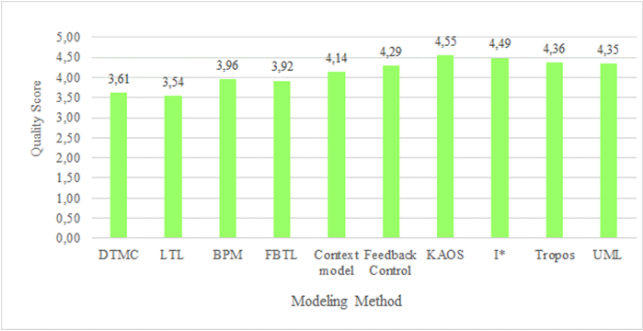


Figure 12. Quality Score of Modeling Methods and Number of Studies

In Table 10 present the quality score of modeling method based on research topic. However, the important note that such goal based models have high score in several research topic but lack in cope with non-funtional requirement issues.

Table 10. Quality score of Modeling Method based on Research Topic

Modeling Method	Cope with Complexity	Cope with Uncertainty	Deal with Adaptation Mechanism	Deal with Non-Functional Requirement
hypergraph grammar			3,01	
fuzzy logic				3,50
LTL	3,54			
Genetic Algorithm		3,67		
propositional logic			3,67	
BPM	4,01			3,33
BNF			3,83	
graph model	3,67	4,00		
FBTL		3,84		4,00
Utility Function				4,00
Block Diagram		4,00		
Algebric semantics	4,00			
DTMC			4,17	3,84
context model	4,34		4,67	3,92
State and Transition System	4,34		4,17	
Feedback Control	3,50	4,50	4,50	4,34
UML	4,67		4,19	4,34
Predicate Logic			4,34	
Organization Model	4,34			
I*		4,67	4,67	3,67
Tropos	4,29	4,67	4,34	
KAOS	4,34	4,53	4,34	
Task Model			4,50	
Markov models		4,67		4,34
Natural Language		4,67		
ASSL		4,67		

4.9 DETAILEDLY DISCUSSED ACTIVITIES

In Table 11 present the quality score of detailedly discussed activities based on view of research topic side. Described that requirement specification is activities that discussed in most research topic as detailed. Software verification, requirement validation and system reconfiguration are the 3 most detailed discussed activities and have high quality score average than others. However, such activities are lack discussed in several research topics.

Table 11. Quality Score of Detailed Discussed Activities based on Research Topic

Activities	Cope with Complexity	Cope with Uncertainty	Deal with Adaptation Mechanism	Deal with Change of Requirement	Deal with Non-Functional Requirement	Deal With Trade-Off Analysis	Dynamic Context	Average of Quality Score
System Evolution		3,67	3,01			3,67		3,45
Derive Design Decision	2,99		4,00					3,50
Requirements Elicitation					4,01		3,67	3,84
Requirements Monitoring	3,61	4,59	3,79		3,67	4,50		4,03
Requirements Evolution		4,00						4,00
Architecture modeling	3,67						4,67	4,17
Requirements modeling	4,28	4,17	4,61	4,67	2,92		4,34	4,16
Requirements Verification	4,34	5,00	4,34	2,83	4,06		4,51	4,18
Adaptation Mechanism		4,67	4,17	4,17		3,51	5,00	4,30
Behavioral modeling							4,33	4,33
Mapping to Architechture	4,34		4,50			3,67	5,00	4,38
Requirements Specification	4,14	4,45	4,56	3,50	4,50	4,67	4,54	4,34
System Reconfiguration		4,67			4,34			4,51
Requirements validation	4,67							4,67
Software Verification			4,67					4,67
Average of Quality Score	4,01	4,40	4,18	3,79	3,91	4,00	4,51	4,17

4.10 THE RELATIONSHIP BETWEEN TOPICS, ACTIVITIES AND MODELING METHODS

The primary studies have been segmented into 15 activities and 7 topics in order to make the analysis of relationships between activities, topics and modeling method. Table 12 presents the relationship between research activities and research topics. Table 13 presents the relationship between modeling method and research topics. The such tables present the relative frequency of each modeling method.

To elaborate how different modeling methods are implemented to a certain research topics and how a modeling method can be adopted into different research topic, table 12 and table 13 respectively can be the references. New topic or new activities can be generated by incorporating uncertainty into the existing topics, since in requirements engineering for self-adaptive systems have made the uncertainty as a first class concept.

Table 12. The Relationship between Topics and Research Activities

	Research Topics							
Research Activities	Cope with Complexity	Cope with Uncertainty	Deal with Adaptation Mechanism	Deal with Change of Requirement	Deal with Non-Functional Requirement	Deal With Trade-Off Analysis	Dynamic Context	Grand Total
Requirements Specification	6	9	6	1	3	1	8	34
Requirements modeling	3	3	3	1	2		3	15
Requirements Monitoring	3	2	4		2	1		12
Requirements Verification	2	1	3	1	3		2	12
Adaptation Mechanism		1	3	1		1	1	7
Mapping to Architechture	2		1			1	1	5
Requirements Elicitation					2		1	3
System Evolution		1	1			1		3
Architecture modeling	1						1	2
Derive Design Decision	1		1					2
System Reconfiguration		1			1			2
Behavioral modeling							1	1
Requirements Evolution		1						1
Requirements validation	1							1
Software Verification			1					1
Grand Total	**19**	**19**	**23**	**4**	**13**	**5**	**18**	**101**

Table 13. The Relationship between Topics and Modeling Methods

Modeling Method	Research Topics							
	Cope with Complexity	Cope with Uncertainty	Deal with Adaptation Mechanism	Deal with Change of Requirement	Deal with Non-Functional Requirement	Deal With Trade-Off Analysis	Dynamic Context	Grand Total
Tropos	4	2	2	1		1	3	13
UML	2		6		2		3	13
I*		2	3	1	2		3	11
KAOS	1	6	1				2	10
Feedback Control	1	1	3	1	1		1	8
context model	1		1		2		2	6
BPM	2				1	1		4
FBTL		2			1		1	4
LTL	4							4
DTMC			1	1	1			3
graph model	1	1						2
Markov models		1			1			2
Predicate Logic			1			1		2
State and Transition System	1		1					2
Algebric semantics	1							1
ASSL		1						1
Block Diagram		1						1
BNF			1					1
BPEL						1		1
Domain specific Model							1	1
first-order logic							1	1
fuzzy logic					1			1
Generic Petri net						1		1
Genetic Algorithm		1						1
hypergraph grammar			1					1
Natural Language		1						1
Organization Model	1							1
propositional logic			1					1
Task Model			1					1
Utility Function					1			1
workflow model							1	1
Grand Total	**19**	**19**	**23**	**4**	**13**	**5**	**18**	**101**

5 CONCLUSION AND FUTURE WORKS

The objective of this chapter is to systematically investigate the research literature of requirements engineering for self-adaptive systems, summarize the research trends, categorize the used modeling methods and requirements engineering activities in research studies published between January 2000 and June 2014. During systematic review process many questions arose. Kitchenham and Charters (2007) provides a set of guidelines for conducting a systematic literature review, which provides steps for formulating research questions to be answered to the review and developing a review protocol (B. Kitchenham & Charters, 2007).

The increasement of publication caused by increased activity of requirement specification since 2007. The increasement was driven by improvement in the model approach such goal based model namely Tropos in (Bresciani et al., 2004), architecture-based self-adaptation called Rainbow in (Garlan et al., 2004) and requirements engineering language namely Relax in (Whittle et al., 2010). Related with venue distribution of publication, from primary studies that collected, there are 60 studies from journals, while 41 papers from conference proceedings. Presented that Lecture Notes in Computer Science (LNCS) is the most popular journal to publish papers on Requirement engineering in self adaptive systems of the studies, while SEAMS is the most prominent conference of the studies. The researchers are listed according to the number of studies included in primary studies among others Betty Cheng, Pete Sawyer, Anna Perini, John Mylopoulos, Andres Ramirez and Nelly Bencomo are active and influential researchers on requirement engineering in self-adaptive systems.

Related with content of studies, from 101 primary studies that collected the 5 activities of research on requirement engineering in self-adaptive systems that most studied, include: requirement specification, requirement modeling, requirement verification, requirement monitoring and adaptation mechanism. Goal-oriented methodologies such as Tropos, i* and KAOS that became most popular requirements modeling methods in the literature. the quality attributes on requirement engineering in self adaptive, include: adaptability, reliability, effectiveness, analysability, understandability, usability and suitability are the most concerned in the selected primary studies. In order to support support the experiment to the research of requirement engineering in self-adaptive systems, mobile application system is application domain that used to the research which deal with specifying requirement on context-awareness system. In the context of service-oriented systems, web service is application domain that illustrate the experiment of the research. In order to support the activities of specification, monitoring and modeling requirements in dynamically adaptive systems (DAS), the researcher most used flood warning system as application domains. Related with the activities that cope with non-functional requirement satisfaction, most researcher used remote data mirroring (RDM) as application domains.

To deal with studies quality, goal based models have high score in several research topic but lack in cope with non-funtional requirement issues. So it takes the development of goal-based models to be able to handle the NFR satisfaction. Requirement specification is activities that discussed in most reseach topic as detailedly. Software verification, requirement validation and system reconfiguration are the 3 most detailedly discussed activities and have high quality score average than others. However, the such activities are lack discussed in several research topics.

The analysis of relationships between activities, topics and modeling method presented that the primary studies have been segmented into 15 activities and 7 topics. New topic or new activities can be generated by incorporating uncertainty into the existing topics, since in requirements engineering for self-adaptive systems have made the uncertainty as a first class concept.

Future work focuses on further are investigating the high-level problem statements in requirements engineering for self-adaptive systems and solution to cope with such problems. Additionally, the relationship between topics and application domain need to be explored.

REFERENCES

Basili, V., Caldiera, G., & Rombach, H. D. (1994). The goal question metric approach. *Encyclopedia of Software Engineering*, *2*, 1–10. Retrieved from http://www.csri.utoronto.ca/~sme/CSC444F/handouts/GQM-paper.pdf

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., & Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, *80*(4), 571–583.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., & Mylopoulos, J. (2004). Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, *8*(3), 203–236.

Brun, Y., Serugendo, G. D. M., & Gacek, C. (2009). Engineering self-adaptive systems through feedback loops. In *Software Engineering for Self-Adaptive Systems* (pp. 48–70). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-02161-9_3

Cheng, B. H. C., Lemos, R. De, & Giese, H. (2009). Software engineering for self-adaptive systems: A research roadmap. *Software Engineering for …*, 1–26. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-02161-9_1

Dobson, S., Zambonelli, F., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., … Schmidt, N. (2006). A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems*, *1*(2), 223–259.

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, *50*(9-10), 833–859.

Ganek, a. G., & Corbi, T. a. (2003). The dawning of the autonomic computing era. *IBM Systems Journal*, *42*(1), 5–18.

Garlan, D., Cheng, S. W., Huang, A. C., Schmerl, B., & Steenkiste, P. (2004). Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, *37*, 46–54.

Kaur, R., & Singh, T. (2010). Analysis and Need of Requirements Engineering. *International Journal of Computer Applications*, *7*(14), 27–32. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.206.2789

Kitchenham, B. A., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering*. Retrieved from http://www.citeulike.org/group/14013/article/7874938

Kitchenham, B., & Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. *Engineering*, *2*, 1051.

Lemos, R. De, Giese, H., & Müller, H. A. (2013). Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II* (pp. 1–32). Springer Berlin Heidelberg.

Macías-Escrivá, F. D., Haber, R., del Toro, R., & Hernandez, V. (2013). Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, *40*(18), 7267–7279.

Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: a roadmap. In *ICSE '00 Proceedings of the Conference on The Future of Software Engineering* (Vol. 1, pp. 35–46). New York, New York, USA: ACM Press. Retrieved from http://portal.acm.org/citation.cfm?doid=336512.336523

Salehie, M., & Tahvildari, L. (2009). Self-adaptive software. *ACM Transactions on Autonomous and Adaptive Systems*, *4*(2), 1–42.

Sawyer, P., Bencomo, N., Whittle, J., Letier, E., & Finkelstein, A. (2010). Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems. In *2010 18th IEEE International Requirements Engineering Conference* (pp. 95–103). IEEE.

Van Lamsweerde, A. (2008). Requirements engineering: From Craft to Discipline. In *ACM SIGSOFT Intl Symp on Foundations of Software Engineering SIGSOFTFSE* (Vol. 164, p. 238). Retrieved from http://dl.acm.org/citation.cfm?id=1453133

Welsh, K., & Sawyer, P. (2010). Understanding the scope of uncertainty in dynamically adaptive systems. In R. Wieringa & A. Persson (Eds.), *Lecture Notes in Computer Science 6182* (Vol. 6182, pp. 2–16). Berlin, Heidelberg: Springer Berlin Heidelberg.

Weyns, D., & Ahmad, T. (2013). Claims and evidence for architecture-based self-adaptation: a systematic literature review. In *Software Architecture* (Vol. 7957, pp. 249–265). Springer Berlin Heidelberg.

Weyns, D., Iftikhar, M. U., Malek, S., & Andersson, J. (2012). Claims and supporting evidence for self-adaptive systems: A literature study. In *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)* (pp. 89–98). IEEE.

Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. C., & Bruel, J. M. (2010). RELAX: A language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, *15*, 177–196.

Yang, Z., Li, Z., Jin, Z., & Chen, Y. (2014). A Systematic Literature Review of Requirements Modeling and Analysis for Self-adaptive Systems. In C. Salinesi & I. Weerd (Eds.), *Requirements Engineering: Foundation for Software Quality* (Vol. 8396, pp. 55–71). Cham: Springer International Publishing.

BIOGRAPHY OF AUTHORS

Slamet Sucipto. Received Bachelor of Computer Science in Informatics Engineering in Mpu Tantular University, Indonesia. He is currently student at Faculty of Informatics Engineering of Eresha school of IT, Indonesia. His current research interest is in field of Software Engineering.

Romi Satria Wahono. Received B.Eng and M.Eng degrees in Computer Science respectively from Saitama University, Japan, and Ph.D in Software Engineering from Universiti Teknikal Malaysia Melaka. He is a lecturer at the Faculty of Computer Science, Dian Nuswantoro University, Indonesia. He is also a founder and chief executive officer of PT Brainmatics Cipta Informatika, a software development company in Indonesia. His current research interests include software engineering and machine learning. Professional member of the ACM and IEEE Computer Society.

Penerapan Teknik Ensemble untuk Menangani Ketidakseimbangan Kelas pada Prediksi Cacat Software

Aries Saifudin
Fakultas Teknik, Universitas Pamulang
aries.saifudin@yahoo.co.id

Romi Satria Wahono
Fakultas Ilmu Komputer, Universitas Dian Nuswantoro
romi@romisatriawahono.net

Abstract: Software berkualitas tinggi adalah software yang tidak ditemukan cacat selama pemeriksaan dan pengujian. Memperbaiki software yang cacat setelah pengiriman membutuhkan biaya jauh lebih mahal dari pada selama pengembangan. Pengujian merupakan proses paling mahal dan menghabiskan waktu sampai 50% dari jadwal pengembangan software. Tetapi belum ada model prediksi cacat software yang berlaku umum. Naïve Bayes merupakan model paling efektif dan efisien, tetapi belum dapat mengklasifikasikan dataset berbasis metrik dengan kinerja terbaik secara umum dan selalu konsisten dalam semua penelitian. Dataset software metrics secara umum bersifat tidak seimbang, hal ini dapat menurunkan kinerja model prediksi cacat software karena cenderung menghasilkan prediksi kelas mayoritas. Secara umum ketidakseimbangan kelas dapat ditangani dengan dua pendekatan, yaitu level data dan level algoritma. Pendekatan level data ditujukan untuk memperbaiki keseimbangan kelas. Sedangkan pendekatan level algoritma dilakukan dengan memperbaiki algoritma atau menggabungkan (*ensemble*) pengklasifikasi tunggal agar menjadi lebih baik. Algoritma *ensemble* yang populer adalah boosting dan bagging. AdaBoost merupakan salah satu algoritma boosting yang telah menunjukkan dapat memperbaiki kinerja pengklasifikasi. Maka pada penelitian ini diusulkan penerapan teknik *ensemble* menggunakan algoritma AdaBoost dan Bagging. Pengklasifikasi yang digunakan adalah Naïve Bayes. Hasil penelitian menunjukkan bahwa teknik *ensemble* dengan algoritma Bagging dapat meningkatkan sensitivitas dan G-Mean secara signifikan, sedangkan AdaBoost tidak dapat meningkatkan secara signifikan. Sehingga disimpulkan bahwa Bagging lebih baik daripada AdaBoost ketika digunakan untuk meningkatkan kinerja Naïve Bayes pada prediksi cacat software.

Keywords: teknik ensemble, ketidakseimbangan kelas, prediksi cacat software

1 PENDAHULUAN

Kualitas software biasanya diukur dari jumlah cacat yang ada pada produk yang dihasilkan (Turhan & Bener, 2007, p. 244). Dengan mengurangi jumlah cacat pada software yang dihasilkan dapat meningkatkan kualitas software. Software berkualitas tinggi adalah software yang tidak ditemukan cacat selama pemeriksaan dan pengujian (McDonald, Musson, & Smith, 2008, p. 6). Pemeriksaan dan pengujian dilakukan terhadap alur dan keluaran dari software. Jumlah cacat yang ditemukan dalam pemeriksaan dan pengujian tidak dapat menjadi satu-satunya ukuran dari kualitas software. Pada banyak kasus, kualitas software lebih banyak dipengaruhi penggunanya, sehingga perlu diukur secara subyektif berdasarkan pada persepsi dan harapan *customer*.

Pengujian merupakan proses pengembangan perangkat lunak yang paling mahal dan banyak memakan waktu, karena sekitar 50% dari jadwal proyek digunakan untuk pengujian (Fakhrahmad & Sami, 2009, p. 206). Software yang cacat menyebabkan biaya pengembangan, perawatan dan estimasi menjadi tinggi, serta menurunkan kualitas software (Gayatri, Nickolas, Reddy, & Chitra, 2009, p. 393). Biaya untuk memperbaiki cacat akibat salah persyaratan (*requirement*) setelah fase penyebaran (*deployment*) dapat mencapai 100 kali, biaya untuk memperbaiki cacat pada tahap desain setelah pengiriman produk mencapai 60 kali, sedangkan biaya untuk memperbaiki cacat pada tahap desain yang ditemukan oleh pelanggan adalah 20 kali (Strangio, 2009, p. 389). Hal ini karena cacat software dapat mengakibatkan *business process* tidak didukung oleh software yang dikembangkan, atau software yang telah selesai dikembangkan harus dilakukan perbaikan atau dikembangkan ulang jika terlalu banyak cacat.

Aktifitas untuk mendukung pengembangan software dan proses menajemen *project* adalah wilayah penelitian yang penting (Lessmann, Baesens, Mues, & Pietsch, 2008, p. 485). Karena pentingnya kesempurnaan software yang dikembangkan, maka diperlukan prosedur pengembangan yang sempurna juga untuk menghindari cacat software. Prosedur yang diperlukan adalah strategi pengujian yang efektif dengan menggunakan sumber daya yang efisien untuk mengurangi biaya pengembangan software.

Prosedur untuk meningkatan kualitas software dapat dilakukan dengan berbagai cara, tetapi pendekatan terbaik adalah pencegahan cacat, karena manfaat dari upaya pencegahannya dapat diterapkan kembali pada masa depan (McDonald, Musson, & Smith, 2008, p. 4). Untuk dapat melakukan pencegahan cacat, maka harus dapat memprediksi kemungkinan terjadinya cacat.

Saat ini prediksi cacat software berfokus pada: 1) memperkirakan jumlah cacat dalam software, 2) menemukan hubungan cacat, dan 3) mengklasifikasikan kerawanan cacat dari komponen software, biasanya ke dalam kelompok rawan dan tidak rawan (Song, Jia, Shepperd, Ying, & Liu, 2011, p. 356). Klasifikasi adalah pendekatan yang populer untuk memprediksi cacat software (Lessmann, Baesens, Mues, & Pietsch, 2008, p. 485) atau untuk mengidentifikasi kegagalan software (Gayatri, Nickolas, Reddy, & Chitra, 2009, p. 393). Untuk melakukan klasifikasi diperlukan data.

Software metrics merupakan data yang dapat digunakan untuk mendeteksi modul software apakah memiliki cacat atau tidak (Chiş, 2008, p. 273). Salah satu metode yang efektif untuk mengidentifikasi modul software dari potensi rawan kegagalan adalah dengan menggunakan teknik *data mining*

yang diterapkan pada *software metrics* yang dikumpulkan selama proses pengembangan software (Khoshgoftaar, Gao, & Seliya, 2010, p. 137). *Software metrics* yang dikumpulkan selama pengembangan disimpan dalam bentuk dataset.

Dataset NASA yang telah tersedia untuk umum merupakan data metrik perangkat lunak yang sangat populer dalam pengembangan model prediksi cacat software, karena 62 penelitian dari 208 penelitian telah menggunakan dataset NASA (Hall, Beecham, Bowes, Gray, & Counsell, 2011, p. 18). Dataset NASA yang tersedia untuk umum telah banyak digunakan sebagai bagian dari penelitian cacat software (Shepperd, Song, Sun, & Mair, 2013, p. 1208). Dataset NASA tersedia dari dua sumber, yaitu NASA MDP (*Metrics Data Program*) *repository* dan PROMISE (*Predictor Models in Software Engineering*) *Repository* (Gray, Bowes, Davey, Sun, & Christianson, 2011, p. 98). Semua dataset dari NASA tersedia secara online, sehingga analisa empiris dapat dengan mudah diulang, diperbaiki, disangkal, dan dievaluasi oleh peneliti lain (Singh & Verma, 2014, pp. 35-36). Menggunakan dataset NASA merupakan pilihan yang terbaik, karena mudah diperoleh dan kinerja dari model yang digunakan menjadi mudah untuk dibandingkan dengan penelitian sebelumnya.

Metode prediksi cacat menggunakan probabilitas dapat menemukan sampai 71% (Menzies, Greenwald, & Frank, 2007, p. 2), lebih baik dari metode yang digunakan oleh industri. Jika dilakukan dengan menganalisa secara manual dapat menemukan sampai 60% (Shull, et al., 2002, p. 254). Hasil tersebut menunjukkan bahwa menggunakan probalitas merupakan metode terbaik untuk menemukan cacat software.

Berdasarkan hasil penelitian yang ada, tidak ditemukan satu metode terbaik yang berguna untuk mengklasifikasikan berbasis metrik secara umum dan selalu konsisten dalam semua penelitian yang berbeda (Lessmann, Baesens, Mues, & Pietsch, 2008, p. 485). Tetapi model Naïve Bayes merupakan salah satu algoritma klasifikasi paling efektif (Tao & Wei-hua, 2010, p. 1) dan efisien (Zhang, Jiang, & Su, 2005, p. 1020), secara umum memiliki kinerja yang baik (Hall, Beecham, Bowes, Gray, & Counsell, 2011, p. 13), serta cukup menarik karena kesederhanaan, keluwesan, ketangguhan dan efektifitasnya (Gayatri, Nickolas, Reddy, & Chitra, 2009, p. 395). Maka dibutuhkan pengembangan prosedur penelitian yang lebih dapat diandalkan sebelum memiliki keyakinan dalam menyimpulkan perbandingan penelitian dari model prediksi cacat software (Myrtveit, Stensrud, & Shepperd, 2005, p. 380). Pengembangan prosedur penelitian dapat dilakukan dengan memperbaiki kualitas data yang digunakan atau dengan memperbaiki model yang digunakan.

Jika dilihat dari *software metrics* yang digunakan, secara umum dataset kualitas software bersifat tidak seimbang (*imbalanced*), karena umumnya cacat dari software ditemukan dalam persentase yang kecil dari modul software (Seiffert, Khoshgoftaar, Hulse, & Folleco, 2011, p. 1). Jumlah dari dataset yang rawan cacat (*fault-prone*) jauh lebih kecil dari pada dataset yang tidak rawan cacat (*nonfault-prone*). Klasifikasi data dengan pembagian kelas yang tidak seimbang dapat menimbulkan penurunan kinerja yang signifikan yang dicapai oleh algoritma belajar (*learning algorithm*) pengklasifikasi standar, yang mengasumsikan distribusi kelas yang relatif seimbang dan biaya kesalahan klasifikasi yang sama (Sun, Mohamed, Wong, & Wang, 2007, p. 3358). Ketepatan parameter tidak dapat digunakan untuk mengevaluasi kinerja dataset yang tidak seimbang (Catal, 2012, p. 195). Membangun model kualitas perangkat lunak tanpa melakukan pengolahan awal terhadap data tidak akan menghasilkan model prediksi cacat software yang efektif, karena jika data yang digunakan tidak seimbang maka hasil prediksi cenderung menghasilkan kelas mayoritas (Khoshgoftaar, Gao, & Seliya, 2010, p. 138). Karena cacat software merupakan kelas minoritas, maka banyak cacat yang tidak dapat ditemukan.

Ada tiga pendekatan untuk menangani dataset tidak seimbang (*unbalanced*), yaitu pendekatan pada level data, level algoritmik, dan menggabungkan atau memasangkan (*ensemble*) metode (Yap, et al., 2014, p. 14). Pendekatan pada level data mencakup berbagai teknik *resampling* dan sintesis data untuk memperbaiki kecondongan distribusi kelas data latih. Pada tingkat algoritmik, metode utamanya adalah menyesuaikan operasi algoritma yang ada untuk membuat pengklasifikasi (*classifier*) agar lebih konduktif terhadap klasifikasi kelas minoritas (Zhang, Liu, Gong, & Jin, 2011, p. 2205). Sedangkan pada pendekatan menggabungkan atau memasangkan (*ensemble*) metode, ada dua algoritma *ensemble-learning* paling populer, yaitu *boosting* dan *bagging* (Yap, et al., 2014, p. 14). Algoritma *boosting* telah dilaporkan sebagai meta-teknik untuk mengatasi masalah ketidakseimbangan kelas (*class imbalance*) (Sun, Mohamed, Wong, & Wang, 2007, p. 3360). Pada pendekatan algoritma dan *ensemble* memiliki tujuan yang sama, yaitu memperbaiki algoritma pengklasifikasi tanpa mengubah data, sehingga dapat dianggap ada 2 pendekatan saja, yaitu pendekatan level data dan pendekatan level algoritma (Peng & Yao, 2010, p. 111). Dengan membagi menjadi 2 pendekatan dapat mempermudah fokus objek perbaikan, pendekatan level data difokuskan pada pengolahan awal data, sedangkan pendekatan level algoritma difokuskan pada perbaikan algoritma atau menggabungkan (*ensemble*).

Bagging dan Boosting telah berhasil meningkatkan akurasi pengklasifikasi tertentu untuk dataset buatan dan yang sebenarnya. Bagging adalah metode *ensemble* yang sederhana namun efektif dan telah diterapkan untuk banyak aplikasi di dunia nyata (Liang & Zhang, 2011, p. 31). Bagging merupakan metode *ensemble* yang banyak diterapkan pada algoritma klasifikasi, dengan tujuan untuk meningkatkan akurasi pengklasifikasi dengan menggabungkan pengklasifikasi tunggal, dan hasilnya lebih baik daripada *random sampling* (Alfaro, Gamez, & Garcia, 2013, p. 1). Secara umum algoritma Boosting lebih baik dari pada Bagging, tetapi tidak merata baik. *Boosting* telah menunjukkan dapat meningkatkan kinerja pengklasifikasi dalam banyak situasi, termasuk ketika data tidak seimbang (Seiffert, Khoshgoftaar, Hulse, & Napolitano, 2008, p. 445). AdaBoost adalah kependekan dari *Adaptive Boosting* (Afza, Farid, & Rahman, 2011, p. 105) (Harrington, 2012, p. 132), merupakan algoritma *machine learning* yang dirumuskan oleh Yoav Freund and Robert Schapire. AdaBoost secara teoritis dapat secara signifikan digunakan untuk mengurangi kesalahan dari beberapa algoritma pembelajaran yang secara konsisten menghasilkan kinerja pengklasifikasi yang lebih baik. AdaBoost diterapkan pada Naïve Bayes dapat meningkatkan kinerja sebesar 33,33% dan memberikan hasil yang akurat dengan mengurangi nilai kesalahan klasifikasi dengan meningkatkan iterasi (Korada, Kumar, & Deekshitulu, 2012, p. 73). Beberapa penelitian tersebut telah menunjukkan bahwa metode *ensemble* (AdaBoost dan Bagging) dapat memperbaiki kinerja pengklasifikasi.

Untuk mencari solusi terbaik terhadap masalah ketidakseimbangan kelas (*class imbalance*) pada prediksi cacat software, maka pada penelitian ini akan diterapkan pendekatan level algoritma, yaitu teknik *ensemble* dengan algoritma AdaBoost dan Bagging. Diharapkan dapat diperoleh model terbaik untuk menyelesaikan masalah ketidakseimbangan

kelas pada prediksi cacat software. Sedangkan algoritma pengklasifikasi yang digunakan adalah Naïve Bayes.

2 PENELITIAN TERKAIT

Penelitian tentang prediksi cacat software telah lama dilakukan, dan sudah banyak hasil penelitian yang dipublikasikan. Sebelum memulai penelitian, perlu dilakukan kajian terhadap penelitian sebelumnya, agar dapat mengetahui metode, data, maupun model yang sudah pernah digunakan. Kajian penelitian sebelumnya ditujukan untuk mengetahui *state of the art* tentang penelitian prediksi cacat software yang membahas tentang ketidakseimbangan (*imbalanced*) kelas.

Penelitian yang dilakukan oleh Riquelme, Ruiz, Rodriguez, dan Moreno (Riquelme, Ruiz, Rodriguez, & Moreno, 2008, pp. 67-74) menyatakan bahwa kebanyakan dataset untuk rekayasa perangkat lunak sangat tidak seimbang (*unbalanced*), sehingga algoritma *data mining* tidak dapat menghasilkan model pengklasifikasi yang optimal untuk memprediksi modul yang cacat. Pada penelitian ini dilakukan analisa terhadap dua teknik penyeimbangan (*balancing*) (yaitu: WEKA *randomly resampling* dan SMOTE) dengan dua algoritma pengklasifikasi umum (J48 dan Naïve Bayes), dan menggunakan lima dataset dari PROMISE *repository*. Hasil penelitian menunjukkan bahwa teknik penyeimbangan (*balancing*) mungkin tidak meningkatkan persentase kasus yang diklasifikasikan dengan benar, tetapi dapat meningkatkan nilai AUC, khususnya menggunakan SMOTE, AUC rata-rata meningkat 11,6%. Hal ini menunjukkan bahwa teknik penyeimbangan (*balancing*) dapat mengklasifikasikan kelas minoritas dengan lebih baik.

Penelitian yang dilakukan oleh Khoshgoftaar, Gao dan Seliya (Khoshgoftaar, Gao, & Seliya, 2010, pp. 137-144) menyatakan bahwa komunitas *data mining* dan *machine learning* sering dihadapkan pada dua masalah utama, yaitu bekerja dengan data tidak seimbang dan memilih fitur terbaik. Penelitian ini menerapkan teknik seleksi fitur untuk memilih atribut penting dan teknik pengambilan data untuk mengatasi ketidakseimbangan kelas. Beberapa sekenario diusulkan menggunakan fitur seleksi dan *resampling* data bersama-sama, yaitu: (1) seleksi fitur berdasarkan data asli, dan pemodelan (prediksi cacat) berdasarkan data asli, (2) seleksi fitur berdasarkan data asli, dan pemodelan berdasarkan data sampel, (3) seleksi fitur berbasis pada data sampel, dan pemodelan berdasarkan data asli, dan (4) seleksi fitur berdasarkan data sampel, dan pemodelan berdasarkan data sampel. Tujuan penelitian ini adalah untuk membandingkan kinerja model prediksi cacat software berdasarkan empat skenario. Pada penelitian ini menggunakan sembilan dataset pengukuran perangkat lunak yang diperoleh dari repositori proyek software PROMISE (*Java-based Eclipse project*). Seleksi fitur menggunakan teknik peringkat fitur (*feature ranking techniques*), *Chi-Square* (CS), *Information Gain* (IG), *Gain Ratio* (GR), dua tipe ReliefF (RF and RFW), dan *Symmetrical Uncertainty* (SU). Metode *resampling* yang digunakan adalah *Random Under-Sampling* (RUS). Sedangkan metode pengklasifikasi yang digunakan adalah *k-Nearest Neighbors* (kNN) dan *Support Vector Machine* (SVM). Hasil empiris menunjukkan bahwa seleksi fitur berdasarkan data sampel menghasilkan lebih baik secara signifikan daripada seleksi fitur berdasarkan data asli, dan bahwa model prediksi cacat melakukan hal yang sama terlepas dari apakah data pelatihan dibentuk menggunakan data sampel atau asli. AUC rata-rata meningkat 1,44% untuk KNN dan 1,04% untuk SVM.

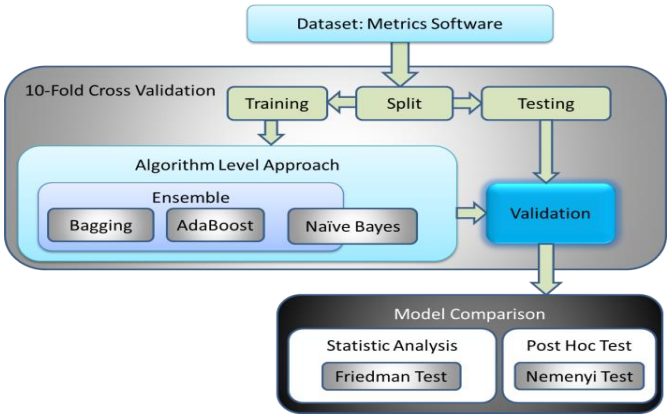
Penelitian yang dilakukan oleh Wahono, Suryana, dan Ahmad (Wahono, Suryana, & Ahmad, 2014, pp. 1324-1333) menyatakan bahwa kinerja model prediksi cacat software berkurang secara signifikan karena dataset yang digunakan mengandung *noise* (kegaduhan) dan ketidakseimbangan kelas. Pada penelitian ini, diusulkan kombinasi metode optimasi metaheuristik dan teknik Bagging untuk meningkatkan kinerja prediksi cacat software. Metode optimasi metaheuristik (algoritma genetik dan *particle swarm optimization*) diterapkan untuk menangani pemilihan fitur, dan teknik Bagging digunakan untuk menangani masalah ketidakseimbangan kelas. Penelitian ini menggunakan 9 NASA MDP dataset dan 10 algoritma pengklasifikasi yang dikelompokkan dalam 5 tipe, yaitu pengklasifikasi statistik tradisional (*Logistic Regression* (LR), *Linear Discriminant Analysis* (LDA), dan Naïve Bayes (NB)), *Nearest Neighbors* (*k-Nearest Neighbor* (k-NN) dan K*), *Neural Network* (*Back Propagation* (BP)), *Support Vector Machine* (SVM), dan *Decision Tree* (C4.5, *Classification and Regression Tree* (CART), dan *Random Forest* (RF)). Hasilnya menunjukkan bahwa metode yang diusulkan dapat memberikan peningkatan yang mengesankan pada kinerja model prediksi untuk sebagian besar pengklasifikasi. Hasil penelitian menunjukkan bahwa tidak ada perbedaan yang signifikan antara optimasi PSO dan algoritma genetik ketika digunakan sebagai seleksi fitur untuk sebagian besar pengklasifikasi pada model prediksi cacat software. AUC rata-rata meningkat 25,99% untuk GA dan 20,41% untuk PSO.

Pada penelitian ini akan menerapkan teknik *ensemble* untuk mengurangi pengaruh ketidakseimbangan kelas dan meningkatkan kemampuan memprediksi kelas minoritas. Teknik *ensemble* yang akan digunakan adalah AdaBoost dan Bagging.

3 METODE YANG DIUSULKAN

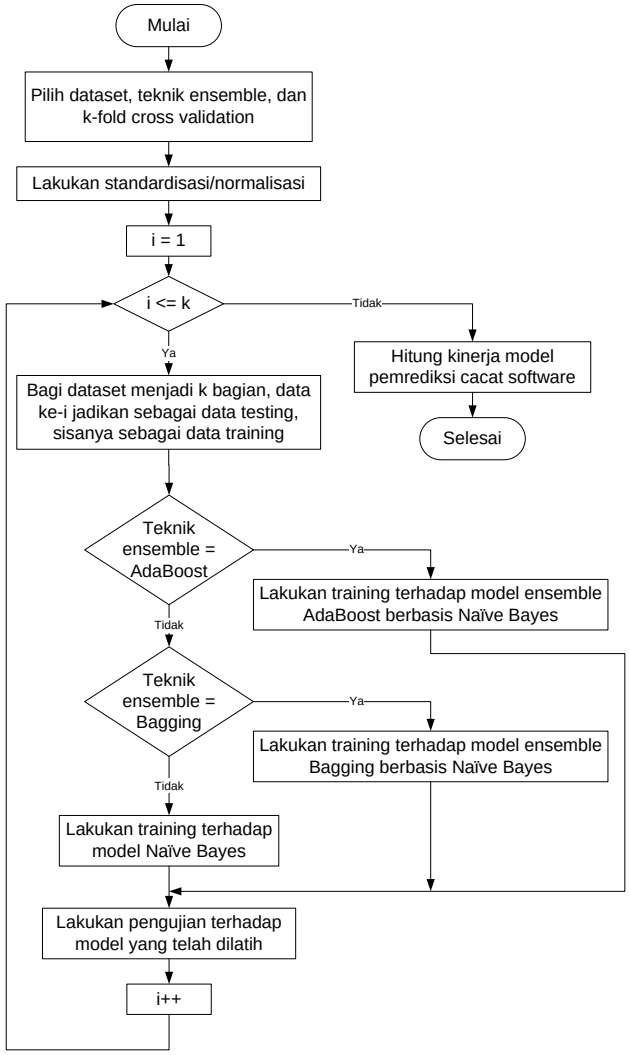
Penelitian ini dilakukan dengan mengusulkan model, mengembangkan aplikasi untuk mengimplementasikan model yang diusulkan, menerapkan pada NASA dataset yang diperoleh dari NASA MDP (*Metrics Data Program*) *repository* dan PROMISE (*Predictor Models in Software Engineering*) *Repository*, dan mengukur kinerjanya. Perangkat lunak yang digunakan untuk mengembangkan aplikasi adalah IDE (*Integrated Development Environment*) NetBeans dengan bahasa Java.

Untuk menangani masalah ketidakseimbangan kelas pada dataset software metrics, diusulkan model menggunakan teknik *ensemble* (AdaBoost, dan Bagging), dan algoritma pengklasifikasi Naïve Bayes. Validasi menggunakan *10-fold cross validation*. Hasil pengukuran dianalisa menggunakan uji Friedman, Nemenyi *post hoc*, dan dibuatkan diagram Demsar. Kerangka kerja model yang diusulkan ditunjukkan pada Gambar 1.



Gambar 1 Kerangka Kerja Model yang Diusulkan

Pada model yang diusulkan, dataset software metrics akan dibagi menjadi 10 bagian. Secara bergantian, setiap bagian secara berurutan dijadikan sebagai data uji, sedangkan bagian lain sebagai data latih. Jika teknik ensemble yang dipilih adalah AdaBoost, maka data latih digunakan untuk melatih model AdaBoost berbasis Naïve Bayes. Jika teknik ensemble yang dipilih adalah Bagging, maka data latih digunakan untuk melatih model Bagging berbasis Naïve Bayes. Model yang telah dilatih diuji dengan data uji, kemudian diukur kinerjanya. Proses ini ditunjukkan dengan flowchart pada Gambar 2.



Gambar 2 Flowchart Model yang Diusulkan

Teknik *ensemble* AdaBoost menggunakan persamaan berikut ini:

$$F(x) = \sum_{t=1}^{T} \propto_t h_t(x)$$

Di mana:
$h_t(x)$: Pengklasifikasi dasar atau lemah
$\propto_t$: Tingkat pembelajaran (*learning rate*)
F(x) : Hasil, berupa pengklasifikasi kuat atau akhir

Algoritma AdaBoost (Zhou & Yu, AdaBoost, 2009, p. 130):

Masukan:
Dataset D = {(x_1,y_1), (x_2,y_2), ..., (x_m,y_m)};
Algoritma pembelajaran lemah (*Weak Learner*) L;
Sebuah *integer* T menyatakan banyaknya iterasi.

Proses:
Inisialisasi berat distribusi:
$D_1(i) = \frac{1}{m}$ untuk semua $i = 1, \dots, m$
for t=1, ... , T:
Melatih pembelajar dasar/lemah h_t dari D menggunakan distribusi D_t
$h_t = L(D,D_t)$
Mengkalkulasi kesalahan dari h_t: $\varepsilon_t = Pr_{x\sim D_{t,y}} I[h_t(\underline{x_i}) \neq y_i]$
if $\varepsilon_t > 0.5$ then break
Menetapkan berat dari h_t: $\propto_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \text{x} \begin{cases} exp(-\propto_t) \; if \; h_t(x_i) = y_i \\ exp(\propto_t) \; if \; h_t(i) \neq y_i \end{cases}$$

Meng-*update* distribusi, di mana Zt adalah faktor normalisasi yang mengaktifkan D_{t+1} menjadi distribusi:

$$\frac{D_t(i) exp\left(-\propto_t y_i h_t(x_i)\right)}{Z_t}$$

end

Keluaran:
Pengklasifikasi akhir/kuat:

$$H(x) = sign(\sum_{t=1}^{T} \propto_t h_t(x))$$

Kesalahan diukur dengan memperhatikan distribusi D_t di mana algoritma pembelajar lemah dilatih. Dalam prakteknya, algoritma pembelajar lemah mungkin merupakan suatu algoritma yang dapat menggunakan bobot D_t pada sampel pelatihan. Atau, bila hal ini tidak memungkinan, bagian dari sampel pelatihan dapat di-*resampling* menurut D_t, dan hasil dari *resampling* yang tidak berbobot (*unweighted*) dapat digunakan untuk melatih algoritma pembelajar yang lemah.
Sedangkan algoritma Bagging adalah:

Masukan: B adalah jumlah *bag*, T adalah data *training* yang berukuran N, x adalah data yang diuji.
Keluaran: Hasil klasifikasi.

Ulangi untuk b = 1, 2, ..., B
a. Buat bootstrap BS_b dengan mengambil sampel dari T sejumlah N dengan penggantian.
b. Latih pengklasifikasi tunggal C_b dengan *boostrap* BS_b

akhir perulangan

Gabungkan hasil pengklasifikasi tunggal C_b (x_i); b = 1, 2, ..., B , hasil klasifikasi yang paling banyak dijadikan keputusan final klasifikasi, mengikuti rumus:

$$C_f(x_i) = \arg max_{j \in Y} \sum_{b=1}^{B} I(C_b(x_i) = j)$$

Bagging adalah metode yang mengkombinasikan *bootstrapping* dan *aggregating* (Alfaro, Gamez, & Garcia, 2013, p. 4). Sampel *bootstrap* ini diperoleh dengan melakukan *resampling* dengan penggantian (*with replacemants*) dari dataset asli sehingga menghasilkan jumlah elemen yang sama dari dataset asli.

Bagging dapat benar-benar berguna untuk membangun pengklasifikasi menjadi lebih baik bila pada pengamatan kumpulan data latih yang terdapat *noise* (kegaduhan) (Alfaro, Gamez, & Garcia, 2013, p. 5). Berdasarkan penelitian, perbaikan yang besar didapat ketika menggunakan 10 *bootstrap*, jika menggunakan lebih dari 25 *bootstrap*, banyak tenaga yang dihabiskan.

Untuk mengukur kinerja digunakan *confusion matrix*. *Confusion matrix* memberikan keputusan yang diperoleh dalam pelatihan dan pengujian (Bramer, 2007, p. 89). *Confusion matrix* memberikan penilaian kinerja klasifikasi berdasarkan objek dengan benar atau salah (Gorunescu, 2011, p. 319). *Confusion matrix* merupakan matrik 2 dimensi yang menggambarkan perbandingan antara hasil prediksi dengan kenyataan.

Untuk data tidak seimbang, akurasi lebih didominasi oleh ketepatan pada data kelas minoritas, maka metrik yang tepat adalah AUC (*Area Under the ROC Curve*), F-Measure, G-Mean, akurasi keseluruhan, dan akurasi untuk kelas minoritas (Zhang & Wang, 2011, p. 85). Akurasi kelas minoritas dapat menggunakan metrik TPrate/recall (sensitivitas). G-Mean dan AUC merupakan evaluasi prediktor yang lebih komprehensif dalam konteks ketidakseimbangan (Wang & Yao, 2013, p. 438).

Rumus-rumus yang digunakan untuk melakukan perhitungannya adalah (Gorunescu, 2011, pp. 320-322):

$$Akurasi = \frac{TP+TN}{TP + TN + FP + FN}$$

$$Sensitivitas = recall = TP_{rate} = \frac{TP}{TP + FN}$$

$$Specificity = TN_{rate} = \frac{TN}{TN + FP}$$

$$FP_{rate} = \frac{FP}{FP + TN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$F - Measure = \frac{(1 + \beta^2)\ x\ recall\ x\ precision}{(\beta\ x\ recall + precision)}$$

$$G - Mean = \sqrt{Sensitivitas\ x\ Specificity}$$

F-Measure adalah metrik evaluasi yang populer untuk masalah ketidakseimbangan. *F-Measure* mengkombinasikan *recall*/sensitivitas dan *precision* sehingga menghasilkan metrik yang efektif untuk pencarian kembali informasi dalam himpunan yang mengandung masalah ketidakseimbangan. *F-Measure* juga bergantung pada faktor β, yaitu parameter yang bernilai dari 0 sampai tak terhingga, dan digunakan untuk mengontrol pengaruh dari *recall* dan *precision* secara terpisah. Ini dapat menunjukkan bahwa ketika β bernilai 0, maka pengaruh *precision* terhadap *F-Measure* berkurang, dan sebaliknya, ketika β bernilai tak terhingga, maka pengaruh *recall* berkurang.

Ketika β bernilai 1, maka *F-Measure* terlihat sebagai integrasi kedua ukuran secara seimbang. Secara prinsip, *F-Measure* merepresentasikan rata-rata harmonis antara *recall* dan *precision*.

$$F - Measure = \frac{2\ x\ recall\ x\ precision}{(\ recall\ +\ precision)}$$

Rata-rata harmonis dari dua angka cenderung lebih dekat dengan yang lebih kecil dari keduanya. Oleh karena itu nilai *F-Measure* yang tinggi menjamin bahwa keduanya dari *recall* dan *precision* bernilai cukup tinggi.

Area Under the ROC (Receiver Operating Characteristic) Curve (AUROC atau AUC) adalah ukuran numerik untuk membedakan kinerja model, dan menunjukkan seberapa sukses dan benar peringkat model dengan memisahkan pengamatan positif dan negatif (Attenberg & Ertekin, 2013, p. 114). AUC menyediakan ukuran tunggal dari kinerja pengklasifikasi untuk menilai model mana yang lebih baik secara rata-rata (López, Fernández, & Herrera, 2014, p. 4). AUC merangkum informasi kinerja pengklasifikasi ke dalam satu angka yang mempermudah perbandingan model ketika tidak ada kurva ROC yang mendominasi (Weiss, 2013, p. 27). AUC adalah cara yang baik untuk mendapatkan nilai kinerja pengklasifikasi secara umum dan untuk membandingkannya dengan pengklasifikasi yang lain (Japkowicz, 2013, p. 202). AUC adalah ukuran kinerja yang populer dalam ketidakseimbangan kelas, nilai AUC yang tinggi menunjukkan kinerja yang lebih baik (Liu & Zhou, 2013, p. 75). Sehingga untuk memilih model mana yang terbaik, dapat dilakukan dengan menganalisa nilai AUC.

AUC dihitung berdasarkan rata-rata perkiraan bidang berbentuk trapesium untuk kurva yang dibuat oleh TP_{rate} dan FP_{rate} (Dubey, Zhou, Wang, Thompson, & Ye, 2014, p. 225). AUC memberikan ukuran kualitas antara nilai positif dan negatif dengan nilai tunggal (Rodriguez, Herraiz, Harrison, Dolado, & Riquelme, 2014, p. 375). Ukuran AUC dihitung sebagai daerah kurva ROC (López, Fernández, & Herrera, 2014, p. 4) (Galar, Fernández, Barrenechea, & Herrera, 2013, p. 3462) menggunakan persamaan:

$$AUC = \frac{1 + TP_{rate} - FP_{rate}}{2}$$

Berdasarkan kerangka kerja model yang diusulkan pada Gambar 1, hasil validasi dan pengukuran kinerja, selanjutnya dilakukan analisa statistik dan uji post hoc. Hanya sedikit penelitian prediksi cacat software yang dilaporkan menggunakan kesimpulan statistik dalam menentukan signifikansi hasil penelitian (Lessmann, Baesens, Mues, & Pietsch, 2008, p. 487). Biasanya hanya dilakukan berdasarkan

pengalaman dan percobaan tanpa menerapkan pengujian statistik secara formal. Hal ini dapat menyesatkan dan bertentangan dengan penelitian lain.

Dalam literatur ada dua jenis uji statistik, yaitu uji parametrik dan uji nonparametrik (García, Fernández, Luengo, & Herrera, 2010, p. 2045). Uji parametrik dapat menggunakan uji T (*T-test*) dan ANOVA (*Analysis of Variance*). Uji nonparametrik dapat menggunakan uji tanda (*sign test*), uji peringkat bertanda Wilcoxon (*Wilcoxon signed-rank test*), uji Friedman (*Friedman test*), dan uji konkordansi (keselarasan) Kendall (*Kendall's coefficient of concordance* atau Kendall's W). Untuk melakukan uji statistik tergantung pada data yang digunakan. Untuk uji parametrik menggunakan data dengan distribusi normal, varians bersifat homogen, data bersifat ratio atau interval, nilai tengah berupa rata-rata. Jika tidak memenuhi syarat untuk uji parametrik, sebaiknya menggunakan uji nonparametrik. Aturan secara umum, lebih baik menggunakan uji parametrik daripada nonparametrik, karena uji parametrik lebih deskriminatif dan kuat (Carver & Nash, 2012, p. 249). Tetapi jika tidak tidak memenuhi syarat, maka disarankan menggunakan uji nonparametrik.

Uji t dan ANOVA tidak cocok digunakan pada penelitian *machine learning* (Demšar, 2006, pp. 6, 10), karena perbedaan kinerja harus terdistribusi normal, untuk memastikan bentuk distribusinya minimal 30 data, hasil pengukuran kinerja *machine learning* sering tidak mencukupi. Berdasarkan sifat data yang dihasilkan *machine learning*, uji nonparametrik sebaiknya lebih diutamakan daripada parametrik. Secara keseluruhan, uji nonparametrik yang diberi nama uji peringkat bertanda Wilcoxon dan uji Friedman cocok untuk menguji model *machine learning* (Demšar, 2006, p. 27). Tetapi beberapa uji perbandingan harus digunakan jika ingin membentuk perbandingan statistik dari hasil eksperimen di antara berbagai algoritma (García, Fernández, Luengo, & Herrera, 2010, p. 2060).

Pada uji statistik mungkin telah dinyatakan bahwa ada perbedaan signifikan pada model yang diuji, tetapi tidak menunjukkan model mana yang memiliki perbedaan signifikan. Untuk mengidentifikasi model mana yang berbeda secara signifikan, maka dapat digunakan uji *post hoc* (Corder & Foreman, 2009, p. 80). Uji *post hoc* dapat membantu peneliti dalam upaya memahami pola yang benar dari rata-rata populasi (Huck, 2012, p. 258). Uji post hoc dilakukan dengan membuat tabel perbandingan, atau visualisasi grafis.

Jika hipotesis nol (H_0) ditolak, maka dapat dilanjutkan dengan melakukan uji *post hoc* (Demšar, 2006, p. 11). Untuk menunjukkan perbedaan secara signifikan, maka digunakan Nemenyi *post hoc*. Nemenyi *post hoc* digunakan untuk menyatakan bahwa dua atau lebih pengklasifikasi memiliki kinerja yang berbeda secara signifikan jika rata-rata peringkatnya memiliki perbedaan setidaknya sebesar *Critical Different* (CD), sesuai persamaan:

$$CD = q_{\propto,\infty,K}\sqrt{\frac{K(K+1)}{12D}}$$

Pada di atas nilai $q_{\propto,\infty,K}$ didasarkan pada *studentized range statistic* untuk derajat kebebasan (*degree of freedom*) bernilai tak terhingga (*infinity*). Sedangkan K adalah jumlah pengklasifikasi, dan D adalah jumlah dataset.

Hasil dari uji Friedman dan Nemenyi *post hoc* selanjutnya disajikan dalam bentuk grafis menggunakan diagram Demsar (Demšar, 2006, p. 16) yang telah dimodifikasi (Lessmann, Baesens, Mues, & Pietsch, 2008, p. 491).

4 HASIL EKSPERIMEN

Eksperimen dilakukan menggunakan sebuah laptop DELL Inspiron 1440 dengan prosesor Pentium® Dual-Core CPU T4500 @ 2.30 GHz, memori (RAM) 4,00 GB, dan sistem operasi Windows 7 Ultimate Service Pack 1 32-bit. Sedangkan perangkat lunak untuk mengembangkan aplikasi adalah IDE (*Integrated Development Environment*) NetBeans menggunakan bahasa Java. Untuk menganalisis hasil pengukuran kinerja digunakan aplikasi IBM SPSS statistic 21 dan XLSTAT versi percobaan (*trial*).

Hasil pengukuran kinerja model ketika diterapkan 20 NASA dataset (10 dari NASA MDP *repository* dan 10 dari PROMISE *repository*) ditunjukkan pada Tabel 1 sampai Tabel 10.

Tabel 1 Akurasi NASA MDP repository

Model	NASA MDP Repository									
	CM1	JM1	KC1	KC3	MC2	PC1	PC2	PC3	PC4	PC5
NB	81,65%	78,87%	74,10%	78,87%	72,58%	88,81%	88,09%	36,75%	85,59%	75,03%
AB+NB	81,65%	78,87%	74,10%	78,87%	73,39%	90,87%	88,37%	84,81%	85,98%	75,03%
BG+NB	81,96%	78,76%	73,84%	78,35%	71,77%	88,22%	76,18%	39,51%	85,04%	74,85%

Tabel 2 Akurasi PROMISE repository

Model	PROMISE Repository									
	CM1	JM1	KC1	KC3	MC2	PC1	PC2	PC3	PC4	PC5
NB	82,61%	78,81%	74,01%	82,10%	72,90%	89,66%	92,51%	47,34%	85,43%	74,73%
AB+NB	83,30%	78,81%	74,01%	84,26%	70,97%	89,66%	92,51%	84,74%	85,04%	74,73%
BG+NB	81,92%	78,86%	73,67%	82,10%	72,26%	88,58%	90,38%	63,66%	85,12%	74,91%

Tabel 3 Sensitifitas NASA MDP repository

Model	NASA MDP Repository									
	CM1	JM1	KC1	KC3	MC2	PC1	PC2	PC3	PC4	PC5
NB	30,95%	19,85%	33,67%	36,11%	38,64%	40,00%	12,50%	90,77%	28,98%	22,27%
AB+NB	30,95%	19,85%	33,67%	36,11%	40,91%	34,55%	12,50%	9,23%	48,30%	22,27%
BG+NB	33,33%	20,60%	34,35%	36,11%	40,91%	41,82%	43,75%	91,54%	35,23%	23,58%

Tabel 4 Sensitifitas PROMISE repository

Model	PROMISE Repository									
	CM1	JM1	KC1	KC3	MC2	PC1	PC2	PC3	PC4	PC5
NB	32,61%	19,73%	33,67%	42,86%	35,29%	35,00%	23,81%	85,14%	28,98%	21,62%
AB+NB	32,61%	19,73%	33,67%	40,48%	35,29%	35,00%	23,81%	14,19%	50,00%	21,62%
BG+NB	30,44%	20,10%	33,67%	42,86%	35,29%	36,67%	33,33%	79,05%	31,82%	23,14%

Tabel 5 F-Measure NASA MDP repository

Model	NASA MDP Repository									
	CM1	JM1	KC1	KC3	MC2	PC1	PC2	PC3	PC4	PC5
NB	0,302	0,282	0,397	0,388	0,500	0,367	0,044	0,262	0,358	0,325
AB+NB	0,302	0,282	0,397	0,388	0,522	0,380	0,045	0,130	0,489	0,325
BG+NB	0,322	0,288	0,399	0,382	0,507	0,365	0,075	0,272	0,395	0,336

Tabel 6 F-Measure PROMISE repository

Model	PROMISE Repository									
	CM1	JM1	KC1	KC3	MC2	PC1	PC2	PC3	PC4	PC5
NB	0,283	0,280	0,396	0,383	0,462	0,307	0,089	0,254	0,355	0,316
AB+NB	0,291	0,280	0,396	0,400	0,444	0,307	0,089	0,163	0,481	0,316
BG+NB	0,262	0,284	0,393	0,383	0,456	0,295	0,097	0,314	0,372	0,333

Tabel 7 G-Mean NASA MDP repository

Model	NASA MDP Repository									
	CM1	JM1	KC1	KC3	MC2	PC1	PC2	PC3	PC4	PC5
NB	0,525	0,433	0,544	0,566	0,594	0,610	0,335	0,514	0,524	0,459
AB+NB	0,525	0,433	0,544	0,566	0,611	0,575	0,336	0,297	0,667	0,459
BG+NB	0,545	0,440	0,547	0,564	0,603	0,621	0,580	0,543	0,573	0,470

Tabel 8 G-Mean PROMISE repository

Model	PROMISE Repository									
	CM1	JM1	KC1	KC3	MC2	PC1	PC2	PC3	PC4	PC5
NB	0,537	0,432	0,543	0,614	0,568	0,572	0,472	0,604	0,523	0,452
AB+NB	0,539	0,432	0,543	0,606	0,559	0,572	0,472	0,363	0,673	0,452
BG+NB	0,517	0,436	0,542	0,614	0,565	0,581	0,552	0,699	0,546	0,467

Tabel 9 AUC NASA MDP repository

Model	NASA MDP Repository									
	CM1	JM1	KC1	KC3	MC2	PC1	PC2	PC3	PC4	PC5
NB	0,600	0,572	0,607	0,624	0,649	0,666	0,512	0,600	0,618	0,584
AB+NB	0,600	0,572	0,607	0,624	0,661	0,652	0,513	0,523	0,702	0,584
BG+NB	0,612	0,574	0,608	0,620	0,648	0,671	0,603	0,619	0,641	0,587

Tabel 10 AUC PROMISE repository

Model	PROMISE Repository									
	CM1	JM1	KC1	KC3	MC2	PC1	PC2	PC3	PC4	PC5
NB	0,605	0,571	0,607	0,654	0,633	0,642	0,587	0,640	0,617	0,580
AB+NB	0,609	0,571	0,607	0,656	0,619	0,642	0,587	0,536	0,703	0,580
BG+NB	0,592	0,572	0,604	0,654	0,628	0,644	0,623	0,705	0,628	0,586

Untuk mengetahui algoritma *ensemble* manakah antara AdaBoost dan Bagging yang dapat meningkatkan kinerja Naïve Bayes menjadi lebih baik pada prediksi cacat software, maka dilakukan uji Friedman, Nemenyi *post hoc*, dan disajikan dalam diagram Demsar yang dimodifikasi. Karena nilai CD (*Critical Difference*) pada Nemenyi *post hoc* dipengaruhi oleh nilai *studentized range statistic* (3,314), jumlah model (K=3), dan jumlah dataset (D=20), maka nilainya dihitung sebagai berikut:

$$CD = q_{\propto,\infty,K}\sqrt{\frac{K(K+1)}{12D}} = 3{,}314\sqrt{\frac{3(3+1)}{12 \cdot 20}}$$

$$CD = 0{,}741032928$$

Untuk mengetahui signifikansi perbedaan di antara keempat model, dilakukan uji Friedman menggunakan aplikasi SPSS. Nilai p-value dari uji Friedman ditunjukkan pada Tabel 11.

Tabel 11 Rekap P-Value pada Uji Friedman

Pengukuran	P-Value	Hasil (α = 0,05)
Akurasi	0,011	Sig., Ada perbedaan nilai di antara model
Sensitivitas	0,002	Sig., Ada perbedaan nilai di antara model
F-Measure	0,225	Not Sig., Tidak ada perbedaan nilai di antara model
G-Mean	0,019	Sig., Ada perbedaan nilai di antara model
AUC	0,073	Not Sig., Tidak ada perbedaan nilai di antara model

Berdasarkan signifikansi dari uji Friedman tersebut, maka hanya pengukuran yang memiliki perbedaan secara signifikan saja yang dibuatkan diagram Demsar yang telah dimodifikasi.

Berikut ini adalah tahapan pembuatan diagram Demsar sesuai hasil pengukuran yang diuji:

A. Akurasi

Data yang diperoleh dari pengukuran akurasi model ditunjukkan pada Tabel 12.

Tabel 12 Hasil Pengukuran Nilai Akurasi

Data Set		Model		
		NB	AB+NB	BG+NB
MDP Repository	CM1	81,65%	81,65%	81,96%
	JM1	78,87%	78,87%	78,76%
	KC1	74,10%	74,10%	73,84%
	KC3	78,87%	78,87%	78,35%
	MC2	72,58%	73,39%	71,77%
	PC1	88,81%	90,87%	88,22%
	PC2	88,09%	88,37%	76,18%
	PC3	36,75%	84,81%	39,51%
	PC4	85,59%	85,98%	85,04%
	PC5	75,03%	75,03%	74,85%
PROMISE Repository	CM1	82,61%	83,30%	81,92%
	JM1	78,81%	78,81%	78,86%
	KC1	74,01%	74,01%	73,67%
	KC3	82,10%	84,26%	82,10%
	MC2	72,90%	70,97%	72,26%
	PC1	89,66%	89,66%	88,58%
	PC2	92,51%	92,51%	90,38%
	PC3	47,34%	84,74%	63,66%
	PC4	85,43%	85,04%	85,12%
	PC5	74,73%	74,73%	74,91%

Uji Friedman dilakukan dengan memberikan peringkat setiap nilai pengukuran. Peringkat diberikan pada model untuk setiap dataset, nilai terbesar diberi peringkat 1, terbesar kedua diberi peringkat 2, dan seterusnya. Hasilnya ditunjukkan pada Tabel 13.

Tabel 13 Peringkat Akurasi Model pada Uji Friedman

Data Set		Model		
		NB	AB+NB	BG+NB
MDP Repository	CM1	2,5	2,5	1
	JM1	1,5	1,5	3
	KC1	1,5	1,5	3
	KC3	1,5	1,5	3
	MC2	2	1	3
	PC1	2	1	3
	PC2	2	1	3
	PC3	3	1	2
	PC4	2	1	3
	PC5	1,5	1,5	3
PROMISE Repository	CM1	2	1	3
	JM1	2,5	2,5	1
	KC1	1,5	1,5	3
	KC3	2,5	1	2,5
	MC2	1	3	2
	PC1	1,5	1,5	3
	PC2	1,5	1,5	3
	PC3	3	1	2
	PC4	1	3	2
	PC5	2,5	2,5	1
Jumlah		38,5	32	49,5
Rata-Rata		1,925	1,6	2,475

Nemenyi *post hoc* dilakukan dengan membuat tabel perbandingan berpasangan, tabel p-value, dan tabel signifikansi perbedaan untuk menunjukkan pasangan model mana yang memiliki perbedaan secara signifikan.

Tabel 14 Perbandingan Berpasangan pada Nemenyi *Post Hoc*

	NB	AB+NB	BG+NB
NB	0	0,325	-0,550
AB+NB	-0,325	0	-0,875
BG+NB	0,550	0,875	0

Untuk menghitung nilai P-value Nemenyi *post hoc* digunakan software XLSTAT. P-value yang bernilai lebih kecil dari 0,05 (nilai α) dicetak lebih tebal, ini menunjukkan bahwa ada perbedaan yang signifikan di antara model pada baris dan kolom yang sesuai.

Tabel 15 P-value pada Nemenyi *Post Hoc*

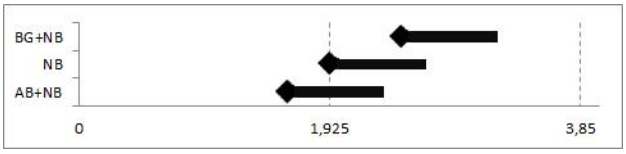
	NB	AB+NB	BG+NB
NB	1	0,559306	0,190612
AB+NB	0,559306	1	**0,015615**
BG+NB	0,190612	**0,015615**	1

Model pada kolom dan baris yang memiliki perbedaan signifikan, diberi nilai Sig. Sedangkan yang perbedaannya tidak signifikan diberi nilai Not.

Tabel 16 Perbedaan Signifikan pada Nemenyi *Post Hoc*

	NB	AB+NB	BG+NB
NB		Not	Not
AB+NB	Not		**Sig**
BG+NB	Not	**Sig**	

Setelah dilakukan uji Friedman dan Nemenyi *post hoc*, selanjutnya ditampilkan pada Gambar 3 menggunakan diagram Demsar yang telah dimodifikasi.



Gambar 3 Perbandingan Akurasi Model Menggunakan Diagram Demsar

Berdasarkan Nemenyi *post hoc* dan diagram Demsar yang dimodifikasi di atas menunjukkan bahwa akurasi model AB+NB dan BG+NB tidak meningkat atau menurun secara signifikan terhadap model NB. Menurut uji Friedman ada perbedaan secara signifikan, perbedaan yang ada adalah antara model AB+NB dengan BG+NB.

B. Sensitivitas

Uji Friedman untuk pengukuran sensitivitas dilakukan dengan cara yang sama pada pengukuran akurasi, dan diperoleh rata-rata peringkat 2,275 untuk NB, 2,275 untuk AB+NB, dan 1,45 untuk BG+NB. Selanjutnya Nemenyi *post hoc* dilakukan dengan membuat tabel perbandingan berpasangan, tabel p-value, dan tabel signifikansi perbedaan untuk menunjukkan pasangan model mana yang memiliki perbedaan secara signifikan.

Tabel 17 Perbandingan Berpasangan pada Nemenyi *Post Hoc*

	NB	AB+NB	BG+NB
NB	0	0	0,825
AB+NB	0	0	0,825
BG+NB	-0,825	-0,825	0

Untuk menghitung nilai P-value Nemenyi *post hoc* digunakan software XLSTAT. P-value yang bernilai lebih kecil dari 0,05 (nilai α) dicetak lebih tebal, ini menunjukkan bahwa ada perbedaan yang signifikan di antara model pada baris dan kolom yang sesuai.

Tabel 18 P-value pada Nemenyi *Post Hoc*

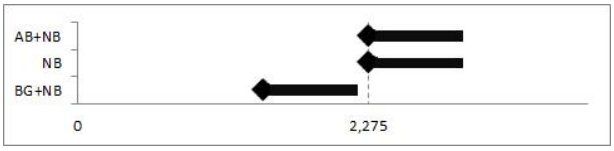
	NB	AB+NB	BG+NB
NB	1	1	**0,024644**
AB+NB	1	1	**0,024644**
BG+NB	**0,024644**	**0,024644**	1

Model pada kolom dan baris yang memiliki perbedaan signifikan, diberi nilai Sig. Sedangkan yang perbedaannya tidak signifikan diberi nilai Not.

Tabel 19 Perbedaan Signifikan pada Nemenyi *Post Hoc*

	NB	AB+NB	BG+NB
NB		Not	**Sig**
AB+NB	Not		**Sig**
BG+NB	**Sig**	**Sig**	

Setelah dilakukan uji Friedman dan Nemenyi *post hoc*, selanjutnya ditampilkan pada Gambar 4 menggunakan diagram Demsar yang telah dimodifikasi.



Gambar 4 Perbandingan Sensitivitas Model Menggunakan Diagram Demsar

Berdasarkan Nemenyi *post hoc* dan diagram Demsar yang dimodifikasi di atas menunjukkan bahwa sensitivitas model BG+NB meningkat secara signifikan dibanding model NB maupun model AB+NB, sedangkan model AB+NB tidak meningkat secara signifikan dibanding model NB.

C. G-Mean

Uji Friedman untuk pengukuran sensitivitas dilakukan dengan cara yang sama pada pengukuran akurasi, dan diperoleh rata-rata peringkat 2,275 untuk NB, 2,2 untuk AB+NB, dan 1,525 untuk BG+NB. Selanjutnya Nemenyi *post hoc* dilakukan dengan membuat tabel perbandingan berpasangan, tabel p-value, dan tabel signifikansi perbedaan untuk menunjukkan pasangan model mana yang memiliki perbedaan secara signifikan.

Tabel 20 Perbandingan Berpasangan pada Nemenyi *Post Hoc*

	NB	AB+NB	BG+NB
NB	0	0,075	0,750
AB+NB	-0,075	0	0,675
BG+NB	-0,750	-0,675	0

Untuk menghitung nilai P-value Nemenyi *post hoc* digunakan software XLSTAT. P-value yang bernilai lebih kecil dari 0,05 (nilai α) dicetak lebih tebal, ini menunjukkan bahwa ada perbedaan yang signifikan di antara model pada baris dan kolom yang sesuai.

Tabel 21 P-value pada Nemenyi *Post Hoc*

	NB	AB+NB	BG+NB
NB	1	0,969467	**0,046560**
AB+NB	0,969467	1	0,082975
BG+NB	**0,046560**	0,082975	1

Model pada kolom dan baris yang memiliki perbedaan signifikan, diberi nilai Sig. Sedangkan yang perbedaannya tidak signifikan diberi nilai Not.

Tabel 22 Perbedaan Signifikan pada Nemenyi *Post Hoc*

	NB	AB+NB	BG+NB
NB		Not	**Sig**
AB+NB	Not		Not
BG+NB	**Sig**	Not	

Setelah dilakukan uji Friedman dan Nemenyi *post hoc*, selanjutnya ditampilkan pada Gambar 5 menggunakan diagram Demsar yang telah dimodifikasi.

Gambar 5 Perbandingan G-Mean Model Menggunakan Diagram Demsar

Berdasarkan Nemenyi *post hoc* dan diagram Demsar yang dimodifikasi di atas menunjukkan bahwa model BG+NB memiliki nilai yang meningkat secara signifikan dibanding NB, sedangkan model AB+NB tidak mengalami peningkatan secara signifikan.

Berdasarkan uji Friedman, Nemenyi *post hoc*, dan diagram Demsar yang dimodifikasi model BG+NB (Bagging berbasis Naïve Bayes) meningkat secara signifikan terhadap model NB untuk pengukuran sensitivitas dan G-Mean, sedangkan model AB+NB (AdaBoost berbasis Naïve Bayes) tidak memberikan peningkatan secara signifikan. Sehingga disimpulkan bahwa model *ensemble* terbaik adalah model BG+NB (Bagging berbasis Naïve Bayes). Hasil penelitian ini bertentangan dengan penelitian yang menyatakan bahwa Naïve Bayes merupakan pembelajar stabil yang tidak dapat ditingkatkan dengan Bagging (Liang, Zhu, & Zhang, 2011, p. 1803), dan AdaBoost dapat meningkatkan kinerja Naïve Bayes (Korada, Kumar, & Deekshitulu, 2012, p. 73). Tetapi menegaskan hasil penelitian yang menyatakan bahwa Boosting tidak bekerja dengan baik pada pengklasifikasi Naïve Bayes, karena merupakan pengklasifikasi yang stabil dengan bias yang kuat (Ting & Zheng , 2003, p. 199).

5 KESIMPULAN

Dua metode *ensemble*, yaitu AdaBoost dan Bagging, telah diusulkan untuk meningkatkan kinerja pengklasifikasi Naïve Bayes dan dilakukan pengukuran kinerjanya. Selanjutnya dilakukan uji statistik terhadap hasil pengukuran menggunakan uji Friedman untuk mengetahui signifikansi perbedaan antarmodel. Untuk pengukuran yang memiliki perbedaan signifikan pada uji Friedman dilakukan Nemenyi *post hoc*, meliputi perbandingan berpasangan, menghitung p-value, dan membuat tabel signifikansi, serta dibuatkan diagram Demsar yang dimodifikasi. Berdasarkan hasil eksperimen yang telah dilakukan menunjukkan bahwa model yang mengintegrasikan Bagging dengan pengklasifikasi Naïve Bayes (BG+NB) memberikan hasil terbaik. Walaupun akurasi secara umum tidak meningkat, tetapi kemampuan memprediksi kelas minoritas meningkat secara signifikan berdasarkan pengukuran sensitivitas dan G-Mean. Sehingga disimpulkan bahwa teknik *ensemble* Bagging lebih baik daripada AdaBoost ketika diintegrasikan dengan pengklasifikasi Naïve Bayes pada prediksi cacat software.

REFERENSI

Afza, A. J., Farid, D. M., & Rahman, C. M. (2011). A Hybrid Classifier using Boosting, Clustering, and Naïve Bayesian Classifier. World of Computer Science and Information Technology Journal (WCSIT), *105-109.*

Alfaro, E., Gamez, M., & Garcia, N. (2013). adabag: An R Package for Classification with Boosting and Bagging. Journal of Statistical Software, *1-35.*

Attenberg, J., & Ertekin, S. (2013). Class Imbalance and Active Learning. In H. He, & Y. Ma, Imbalanced Learning: Foundations, Algorithms, and Applications *(pp. 101-149). New Jersey: John Wiley & Sons.*

Bramer, M. (2007). Principles of Data Mining. *London: Springer.*

Carver, R. H., & Nash, J. G. (2012). Doing Data Analysis with SPSS® Version 18. *Boston: Cengage Learning.*

Catal, C. (2012). Performance Evaluation Metrics for Software Fault Prediction Studies. Acta Polytechnica Hungarica, *9(4), 193-206.*

Chiş, M. (2008). Evolutionary Decision Trees and Software Metrics for Module Defects Identification. World Academy of Science, Engineering and Technology, *273-277.*

Corder, G. W., & Foreman, D. I. (2009). Nonparametric Statistics for Non-statisticians: A Step-by-step Approach. *New Jersey: John Wiley & Sons.*

Demšar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research, *1–30.*

Dubey, R., Zhou, J., Wang, Y., Thompson, P. M., & Ye, J. (2014). Analysis of Sampling Techniques for Imbalanced Data: An n = 648 ADNI Study. NeuroImage, *220–241.*

Fakhrahmad, S. M., & Sami, A. (2009). Effective Estimation of Modules' Metrics in Software Defect Prediction. Proceedings of the World Congress on Engineering *(pp. 206-211). London: Newswood Limited.*

Galar, M., Fernández, A., Barrenechea, E., & Herrera, F. (2013). EUSBoost: Enhancing Ensembles for Highly Imbalanced Data-sets by Evolutionary Under Sampling. Pattern Recognition, *3460–3471.*

García, S., Fernández, A., Luengo, J., & Herrera, F. (2010). Advanced Nonparametric Tests for Multiple Comparisons in the Design of Experiments in Computational Intelligence and Data Mining: Experimental Analysis of Power. Information Sciences, *2044–2064.*

Gayatri, N., Nickolas, S., Reddy, A., & Chitra, R. (2009). Performance Analysis Of Data Mining Algorithms for Software Quality Prediction. International Conference on Advances in Recent Technologies in Communication and Computing *(pp. 393-395). Kottayam: IEEE Computer Society.*

Gorunescu, F. (2011). Data Mining: Concepts, Models and Techniques. *Berlin: Springer-Verlag.*

Gray, D., Bowes, D., Davey, N., Sun, Y., & Christianson, B. (2011). The Misuse of the NASA Metrics Data Program Data Sets for Automated Software Defect Prediction. Evaluation & Assessment in Software Engineering (EASE 2011), 15th Annual Conference on, *(pp. 96-103). Durham.*

Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2011). A Systematic Literature Review on Fault Prediction Performance in Software Engineering. IEEE Transactions on Software Engineering, Accepted for publication - available online, *1-31.*

Harrington, P. (2012). Machine Learning in Action. *New York: Manning Publications Co.*

Huck, S. W. (2012). Reading Statistics and Research. *Boston: Pearson Education.*

Japkowicz, N. (2013). Assessment Metrics for Imbalanced Learning. In H. He, & Y. Ma, Imbalanced Learning: Foundations, Algorithms, and Applications *(pp. 187-206). New Jersey: John Wiley & Sons.*

Khoshgoftaar, T. M., Gao, K., & Seliya, N. (2010). Attribute Selection and Imbalanced Data: Problems in Software Defect Prediction. International Conference on Tools with Artificial Intelligence *(pp. 137-144). IEEE Computer Society.*

Korada, N. K., Kumar, N. P., & Deekshitulu, Y. (2012). Implementation of Naïve Bayesian Classifier and Ada-Boost

Algorithm Using Maize Expert System. International Journal of Information Sciences and Techniques (IJIST) Vol.2, No.3, *63-75.*

Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. IEEE Transactions on Software Engineering, *485-496.*

Liang, G., & Zhang, C. (2011). Empirical Study of Bagging Predictors on Medical Data. Proceedings of the 9-th Australasian Data Mining Conference (AusDM'11) *(pp. 31-40). Ballarat, Australia: Australian Computer Society, Inc.*

Liang, G., Zhu, X., & Zhang, C. (2011). An Empirical Study of Bagging Predictors for Different Learning Algorithms. Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence *(pp. 1802-1803). California: AAAI Press.*

Liu, X.-Y., & Zhou, Z.-H. (2013). Ensemble Methods for Class Imbalance Learning. In H. He, & Y. Ma, Imbalanced Learning: Foundations, Algorithms, and Applications *(pp. 61-82). New Jersey: John Wiley & Sons.*

López, V., Fernández, A., & Herrera, F. (2014). On the Importance of the Validation Technique for Classification with Imbalanced Datasets: Addressing Covariate Shift when Data is Skewed. Information Sciences, *1–13. doi:10.1016/j.ins.2013.09.038*

McDonald, M., Musson, R., & Smith, R. (2008). The Practical Guide to Defect Prevention. *Washington: Microsoft Press.*

Menzies, T., Greenwald, J., & Frank, A. (2007). Data Mining Static Code Attributes to Learn Defect Predictors. IEEE Transactions on Software Engineering, *1-12.*

Myrtveit, I., Stensrud, E., & Shepperd, M. (2005). Reliability and Validity in Comparative Studies of Software Prediction Models. IEEE Transactions on Software Engineering, *380-391.*

Peng, Y., & Yao, J. (2010). AdaOUBoost: Adaptive Over-sampling and Under-sampling to Boost the Concept Learning in Large Scale Imbalanced Data Sets. Proceedings of the international conference on Multimedia information retrieval *(pp. 111-118). Philadelphia, Pennsylvania, USA: ACM.*

Riquelme, J. C., Ruiz, R., Rodriguez, D., & Moreno, J. (2008). Finding Defective Modules From Highly Unbalanced Datasets. Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos *(pp. 67-74). Gijon, España: SISTEDES.*

Rodriguez, D., Herraiz, I., Harrison, R., Dolado, J., & Riquelme, J. C. (2014). Preliminary Comparison of Techniques for Dealing with Imbalance in Software Defect Prediction. 18th International Conference on Evaluation and Assessment in Software Engineering (EASE 2014) *(pp. 371-380). New York: ACM. doi:10.1145/2601248.2601294*

Seiffert, C., Khoshgoftaar, T. M., Hulse, J. V., & Folleco, A. (2011). An Empirical Study of the Classification Performance of Learners on Imbalanced and Noisy Software Quality Data. Information Sciences, *1-25.*

Seiffert, C., Khoshgoftaar, T. M., Hulse, J. V., & Napolitano, A. (2008). Resampling or Reweighting: A Comparison of Boosting Implementations. 20th IEEE International Conference on Tools with Artificial Intelligence, *445-451.*

Shepperd, M., Song, Q., Sun, Z., & Mair, C. (2013). Data Quality: Some Comments on the NASA Software Defect Data Sets. IEEE Transactions on Software Engineering, *1208-1215. doi:10.1109/TSE.2013.11*

Shull, F., Basili, V., Boehm, B., Brown, A. W., Costa, P., Lindvall, M., . . . Zelkowitz, M. (2002). What We Have Learned About Fighting Defects. METRICS '02 Proceedings of the 8th International Symposium on Software Metrics *(pp. 249-258). Washington: IEEE Computer Society.*

Singh, P., & Verma, S. (2014). An Efficient Software Fault Prediction Model using Cluster Based Classification. International Journal of Applied Information Systems (IJAIS), *7(3), 35-41.*

Song, Q., Jia, Z., Shepperd, M., Ying, S., & Liu, J. (2011). A General Software Defect-Proneness Prediction Framework. IEEE Transactions on Software Engineering, *356-370.*

Strangio, M. A. (2009). Recent Advances in Technologies. *Vukovar: In-Teh.*

Sun, Y., Mohamed, K. S., Wong, A. K., & Wang, Y. (2007). Cost-sensitive Boosting for Classification of Imbalanced Data. Pattern Recognition Society, *3358-3378.*

Tao, W., & Wei-hua, L. (2010). Naïve Bayes Software Defect Prediction Model. Computational Intelligence and Software Engineering (CiSE) *(pp. 1-4). Wuhan: IEEE Computer Society. doi:10.1109/CISE.2010.5677057*

Ting, K. M., & Zheng , Z. (2003). A Study of AdaBoost with Naive Bayesian Classifiers: Weakness and Improvement. Computational Intelligence, 19*(2), 186-200. doi:10.1111/1467-8640.00219*

Turhan, B., & Bener, A. (2007). Software Defect Prediction: Heuristics for Weighted Naive Bayes. Proceedings of the 2nd International Conference on Software and Data Technologies (ICSOFT'07), *(pp. 244-249).*

Wahono, R. S., Suryana, N., & Ahmad, S. (2014, May). Metaheuristic Optimization based Feature Selection for Software Defect Prediction. Journal of Software, 9*(5), 1324-1333. doi:10.4304/jsw.9.5.1324-1333*

Wang, S., & Yao, X. (2013). Using Class Imbalance Learning for Software Defect Prediction. IEEE Transactions on Reliability, *434-443.*

Weiss, G. M. (2013). Foundations of Imbalanced Learning. In H. He, & Y. Ma, Imbalanced Learning: Foundations, Algorithms, and Applications *(pp. 13-41). New Jersey: John Wiley & Sons.*

Yap, B. W., Rani, K. A., Rahman, H. A., Fong, S., Khairudin, Z., & Abdullah, N. N. (2014). An Application of Oversampling, Undersampling, Bagging and Boosting in Handling Imbalanced Datasets. Proceedings of the First International Conference on Advanced Data and Information Engineering (DaEng-2013). 285*, pp. 13-22. Singapore: Springer. doi:10.1007/978-981-4585-18-7_2*

Zhang, D., Liu, W., Gong, X., & Jin, H. (2011). A Novel Improved SMOTE Resampling Algorithm Based on Fractal. Computational Information Systems, *2204-2211.*

Zhang, H., & Wang, Z. (2011). A Normal Distribution-Based Over-Sampling Approach to Imbalanced Data Classification. Advanced Data Mining and Applications - 7th International Conference *(pp. 83-96). Beijing: Springer.*

Zhang, H., Jiang, L., & Su, J. (2005). Augmenting Naïve Bayes for Ranking. ICML '05 Proceedings of the 22nd international conference on Machine learning *(pp. 1020 - 1027). New York: ACM Press. doi:http://dx.doi.org/10.1145/1102351.1102480*

Zhou, Z.-H., & Yu, Y. (2009). The Top Ten Algorithms in Data Mining. *(X. Wu, & V. Kumar, Eds.) Florida: Chapman & Hall/CRC.*

BIOGRAFI PENULIS

Aries Saifudin. Memperoleh gelar A.Md. di bidang Teknik Elektronika dari Politeknik Universitas Brawijaya, Malang, gelar S.T. di bidang Teknik Informatika dari Universitas Mercu Buana, Jakarta, dan gelar M.Kom di bidang Rekayasa Perangkat Lunak dari STMIK ERESHA, Jakarta. Dia sebagai dosen tetap di Universitas Pamulang. Minat penelitiannya saat ini meliputi rekayasa perangkat lunak dan machine learning.

Romi Satria Wahono. Memperoleh gelar B.Eng dan M.Eng pada bidang ilmu komputer di Saitama University Japan, dan Ph.D pada bidang software engineering di Universiti Teknikal Malaysia Melaka. Pengajar dan peneliti di Fakultas Ilmu Komputer, Universitas Dian Nuswantoro. Pendiri dan CEO PT Brainmatics, perusahaan yang bergerak di bidang pengembangan software. Minat penelitian pada bidang software engineering dan machine learning. Profesional member dari asosiai ilmiah ACM, PMI dan IEEE Computer Society.

Absolute Correlation Weighted Naïve Bayes for Software Defect Prediction

Rizky Tri Asmono, Romi Satria Wahono and Abdul Syukur
Faculty of Computer Science, Dian Nuswantoro University, Semarang, Indonesia
rtriasmono@gmail.com, romi@romisatriawahono.net, abdul_s@dosen.dinus.ac.id

Abstract: The maintenance phase of the software project can be very expensive for the developer team and harmful to the users because some flawed software modules. It can be avoided by detecting defects as early as possible. Software defect prediction will provide an opportunity for the developer team to test modules or files that have a high probability defect. Naïve Bayes has been used to predict software defects. However, Naive Bayes assumes all attributes are equally important and are not related each other while, in fact, this assumption is not true in many cases. Absolute value of correlation coefficient has been proposed as weighting method to overcome Naïve Bayes assumptions. In this study, Absolute Correlation Weighted Naïve Bayes have been proposed. The results Wilcoxon signed-rank test on experiment results show that the proposed method improves the performance of Naïve Bayes for classifying defect-prone on software defect prediction.

Keywords: Software Defect Prediction, Naïve Bayes, Absolute Correlation Weighted Naïve Bayes, Correlation Coefficient, Absolute Correlation

1 INTRODUCTION

Software is computer programs and related documentation. Software products can be expanded for a specific customer or may be developed for the common marketplace in accordance with the functions and needs. Develop a flawless software is difficult and often times there are some errors or bugs unknown or unexpected defects, although the software-development methodology has been applied with cautious(Okutan & Yıldız, 2012). The maintenance phase of the software project will be very expensive for the developer team and harmful to the users because some flawed software modules. Surely, it can be avoided by detecting defects as early as possible. Defect prediction will provide an opportunity for the developer team to test modules or files that have a high probability defect. The completion of defect prediction problems currently focusing on 1) estimate the number of defects in the existing software systems, 2) discovering defect associations and 3) classification on the defect-prone of software, specially defect and non-defect label(Song, Jia, Shepperd, Ying, & Liu, 2011). The things that detrimental to users and developer team can be avoided as early as possible with a software defect prediction.

For classifying defect-prone, Hall conducted an investigation on software defect prediction(T. Hall, Beecham, Bowes, Gray, & Counsell, 2012). Hall compared Decision Tree, Logistic Regression, Naïve Bayes, Neural Network, C4.5 etc. The results of the investigation showed the two best methods that can be used to predict software defects are Naive Bayes (NB) and Logistic Regression. Logistic Regression is a statistical probabilistic classification method. The advantages of logistic regression are computationally inexpensive, slight to implement and mild to interpret knowledge representation. The disadvantages of logistic regression are prone to under fitting and may have a low accuracy(Harrington, 2012). Naïve Bayes is a modest probabilistic classifier. It is very comfortable because it does not require any complicated parameter estimation. Therefore, Naive Bayes ready to be used for large amounts of data. Moreover, Naive Bayes is also very facile to explain so the users who do not have the technological classification capability can understand the reason why the classification was made(X. Wu & Kumar, 2009). However, Naive Bayes assumes all attributes are equally important and are not related each other while, in fact, this assumption is not true in many cases(J. Wu & Cai, 2011), (Turhan & Bener, 2009), (Liangxiao Jiang, 2011). The assumption made by Naive Bayes can be detrimental to its performance in real data mining applications.

Naïve Bayes assumes that all the attributes are not dependent on each other, in fact, the class depends on others attribute. Naïve Bayes also assumes the relationship between class and one attribute as strong as the relationship between class and other attribute(Turhan & Bener, 2009). The case mentioned previously clearly unrealistic. For example, data set for evaluate risk of loan application, it seems not fair to assume that between income, age and education levels are equally important. The assumption made by Naïve Bayes harming the performance of classification in reality(Webb, Boughton, & Wang, 2005). This assumption can cause the unwanted error increase.

Many methods have been developed to cover this attribute independence assumption. Jiang(Liangxiao Jiang, Wang, Cai, & Yan, 2007), (L. Jiang, Cai, & Wang, 2010) categorizes solutions to these problems into five: 1) Attribute selection, 2) Local Learning, 3) Attribute Weighting, 4) Instance Weighting and 5) Structure Extension. Previous researchers have proposed many useful methods to evaluate the important attributes. Ratanamahatana use Decision Tree as feature selection on Naïve Bayes(Ratanamahatana & Gunopulos, 2003). Zhang use Gain Ratio to determine attribute weight on Naïve Bayes(Zhang, 2004). Wu use Differential Evolution Algorithm to weighting attribute(J. Wu & Cai, 2011). Decision Tree-based attribute weighting for Naïve Bayes proposed by Hall(M. Hall, 2007). Averaged n-Dependence Estimators (AnDE) was proposed by Webb(Webb, Boughton, Zheng, Ting, & Salem, 2011). AnDE was developed from Averaged One-Dependence estimators (Aode) which reduce the Naive Bayes independence assumption(Webb et al., 2005). Zaidi proposed Weighting attributes to Alleviate Naive Bayes Independence Assumption (WANBIA) by set all weights to a single value(Zaidi, Cerquides, Carman, & Webb, 2013). Taheri proposed Attribute Weighted Naive Bayes (AWNB) which define more than one weight for each attribute(Taheri, Yearwood, Mammadov, & Seifollahi, 2013). AWNB limited to binary classification.

Because of the attribute does not have the same role, some of them more important than the others, one of the ways to develop Naïve Bayes is set a different weight value of each attributes. It is becoming the main idea of the new algorithm called Weighting Naive Bayes, abbreviated WNB, weight value depending on how significant these attributes in the probability, more influential an attribute in probabilities, higher the weight value. For weighting the attribute, this study using correlation coefficients to measure relevancy between class and attributes. Arauzo-Azora has been conducting an empirical study of feature selection method(Arauzo-Azofra, Aznarte, & Benítez, 2011). The study evaluated a broad overview of feature selection methods. For weighting attributes by calculating the relevance between attributes get the highest average rank from all the ways for Naïve Bayes. Correlation coefficient can be used to measure the relevance between attributes(Golub, 1999), (Furey et al., 2000), (Pavlidis, Weston, Cai, & Grundy, 2001). It used to measure the strength of relationship between two attributes(Freund & Wilson, 2003). The value of the correlation coefficients is between +1 and -1. Value of +1 and -1 indicate positive and negative relationship. Because it only requires the strength of the relationship between the attributes then absolute value of the correlation coefficients is used.

In this study, Absolute Correlation Weighted Naïve Bayes has been proposed. Absolute correlation is used as a weight because it shows how strong relevance between attributes. The purpose of this study is to improve the performance of Naïve Bayes for classifying defect-prone on software defect prediction.

2 RELATED WORK

While many studies, including individual study report the performance comparison of modeling techniques, there is no explicit consensus appear that conduct best when distinctive studies that looked at in isolation(T. Hall et al., 2012). Mizuno and Kikuno(Mizuno & Kikuno, 2007) reported, the techniques they learned, Orthogonal Sparse Bigrams Markov models (OSB) are most fit for the defect prediction. Bibi et al.(Bibi, Tsoumakas, Stamelos, & Vlahvas, 2006) reported that Regression via Classification (RVC) works fine. Khoshgoftaar et al.(Khoshgoftaar, Yuan, Allen, Jones, & Hudepohl, 2002) reported that the defect-prone modules predicted as uncertain, can be effectively classified using Tree Disc (TD) technique. Khoshgoftaar and Seliya(Khoshgoftaar & Seliya, 2004) also reported that the Case-Based Reasoning (CBR) did not predict well with C4.5 as well under performing. Arisholm et al.(Arisholm, Briand, & Johannessen, 2010) reported that their comprehensive performance comparison showed there is no difference between predictive modeling techniques they investigated.

A clearer picture appears to arise from the detailed analysis conducted by Hall(T. Hall et al., 2012) on the performance of the model. Hall(T. Hall et al., 2012) findings indicate that actually performance can be associated with modeling techniques that is used. Their comparative analysis showed that studies using Support Vector Machine (SVM) technique appear less well. It probably performed poorly because they need the optimization of parameters where it is uncommon done in the study of defect prediction for best performance(Hsu, Chang, & Lin, 2003). C4.5 model apparently poor performing if they use imbalanced data (Arisholm, Briand, & Fuglerud, 2007; Arisholm et al., 2010). The comparative analysis has been done by Hall also shows that the model performs proportionately correctly are comparatively easy technique that simple to use and rightly understood. Logistic Regression and Naïve Bayes, specifically, appears to be technique that is used in the model are performing relatively well. The model appears to work correctly when the proper techniques have been appropriate for the dataset.

Lin et al.(Lin & Yu, 2011) have a research problem that Naïve Bayes assumption that assumes the value of attributes are independent with other attributes. Lin et al.(Lin & Yu, 2011) proposed PSO-based Weighted Naïve Bayesian Classifier to mitigate this assumption. Particle Swarm Optimization algorithm applied on a weighted naive Bayes classification, through the automatic search behavior of particles, to avoid the mistakes of other methods. The value of the search results by the swarm is used as a weight to each attribute. The research results show that the correct rate-based Weighted Naïve Bayesian Classifier higher than the Naïve Bayes.

Taheri et al.(Taheri et al., 2013) have a research problem that attributes independence assumption created by NB classifier adverse classification performance when, in fact, infringed. On research, Taheri et al.(Taheri et al., 2013) proposed a new attribute weighted Naïve Bayes classifier, called AWNB, which provide more than one weight for every attribute. The results presented show that the accuracy of the proposed method far better compared to Naïve Bayes in every data set(Taheri et al., 2013). It also indicates greater accuracy from AWNB, in general, compared with the results obtained by INB and TAN.

The research issue of Wu et al.(J. Wu & Cai, 2011) is independence assumption made by Naive Bayes that all attributes not related to one another adverse classification performance when it is infringed in reality. In order to weaken the assumption of independent attributes, Wu et al.(J. Wu & Cai, 2011) suggested Different Evolution as a weighting method on Weighted Naïve Bayes. Wu et al.(J. Wu & Cai, 2011) comparing Different Evolution with some other weighting methods and the performance of the Different Evolution is better than other weighting methods.

Naïve Bayes is easy to build, because it has a very simple structure(X. Wu & Kumar, 2009). Learning Naïve Bayes only involves teaching the probability table, to be specific, the conditional probability tables for each attribute, from training examples. This means, the values of the probability $p\ (a_i|c)$ must be determined from the training sample, for each value a_i of attribute A_i considering the value of the variable c on class C.

In Naïve Bayes, it is assumed that the attributes are independent one another provided class(J. Wu & Cai, 2011), (Turhan & Bener, 2009), (L. Jiang et al., 2010). Each attribute only has class variables as its parent(J. Wu & Cai, 2011), (Liangxiao Jiang et al., 2007), $P\ (E|c)$ is calculated by:

$$p(E|c) = p(a_1, a_2, \dots, a_n|c) = \prod_{i=1}^{n} p(a_i|c)$$

Where $p\ (a_i|c)$ is referred the likelihood of A_i, and the instance $E = (a_1, a_2, \dots, a_n)$.

Due to each sample E the value of $p\ (E)$ is constant. The possibility of the establishment of a class label for an example is:

$$p(c|E) = p(c) \prod_{i=1}^{n} p(a_i|c)$$

Example E are classified into the class $C = c'$ if and only if

$$p(c'|E) = \arg\max_{c} p(c|E)$$

More exactly, the classification of which granted by Naïve Bayes, denoted by $V_{nb}(E)$, is defined as follows

$$V_{nb}(E) = \arg\max_{c} p(c) \prod_{i=1}^{n} p(a_i|c)$$

Because of the conditional independence assumption uncommon properly, in reality, it is reasonable to extend the Naïve Bayes to relax the assumption of conditional independence. There are two primary ways to relax the assumption (Zhang, 2004). First, Naïve Bayes structure is extended to explicitly represent dependencies between attributes, and generated model referred to augmented Naïve Bayes (ANB)(Friedman, Geiger, & Goldszmidt, 1997). Second, Attributes are weighted differently, and resultant model is called Weighted Naïve Bayes (WNB). Weighted Naïve Bayes is formally defined as follows.

$$V_{wnb}(E) = \arg\max_{c} p\,(c) \prod_{i=1}^{n} p(a_i|c)^{w_i}$$

Where $V_{wnb}(E)$ shows the classification provided by Weighted Naïve Bayes, and w_i is weight of attribute A_i.

3 PROPOSED METHOD

The correlation coefficient measures the power of linear relationship between two quantitative variables, typically a ratio or interval(Freund & Wilson, 2003). The correlation coefficient has the properties, which are 1) its value is among +1 and -1 inclusively, 2) the values +1 and -1 indicate a positive relationship and negative exact, respectively, among the variables, 3) a correlation of zero shows there is no linear relationship exists between two variables and 4) the correlation coefficient is symmetric to x and y. Thus the size of the power of the linear relationship irrespective of whether x or y is the independent variable.

The correlation coefficient can be defined as follows

$$r = \frac{\sum(x - \bar{x})\,(y - \bar{y})}{\sum\,(x - \bar{x})^2 \sum\,(y - \bar{y})^2}$$

Where r symbolize correlation coefficient, $\bar{x}$ symbolize mean value of x and $\bar{y}$ symbolize mean value of y.

Various correlation coefficients are used as a weighting method(Guyon, Weston, Barnhill, & Vapnik, 2002). The coefficient used in Golub(Golub, 1999) is defined as

$$w_i = \frac{(\mu_i(+) - \mu_i(-))}{(\sigma_i(+) + \sigma_i(-))}$$

Where μ_i and σ_i are the mean and standard deviation of the values of attribute i for all of class $(+)$ or class$(-), i = 1, 2, \ldots, n$, w_i large positive value shows a strong correlation with class $(+)$ while the large negative value of w_i show a strong correlation with class $(-)$. Other people(Furey et al., 2000) have used the absolute value of w_i as a weighting method. Recently, Pavlidis(Pavlidis et al., 2001) has been using the associated coefficients which defined below

$$w_i = \frac{(\mu_i(+) - \mu_i(-))^2}{(\sigma_i(+)^2 + \sigma_i(-)^2)}$$

Zhang(Zhang, 2004) extended the Naïve Bayes to relax the assumption of conditional independence. Zhang(Zhang, 2004) performed Weighted Naïve Bayes. Correlation coefficient is used as a weight because it shows how strong relevance between attributes. Based on the coefficient that used in Golub(Golub, 1999), the proposed method uses the absolute correlation coefficient because it only requires the strength of the relationship between the attributes. This idea is similar with Furey et al.(Furey et al., 2000) that can be defined as follows.

$$w_i = \left|\frac{(\mu_{ij} - \mu_{i\bar{j}})}{(\sigma_{ij} + \sigma_{i\bar{j}})}\right|$$

Where w_i is a weight of attribute i, μ_{ij} is mean values of attribute i for class j, $\mu_{i\bar{j}}$ is mean values of attribute i for class non j, σ_{ij} is standard deviation of the values of attribute i for class j and $\sigma_{i\bar{j}}$ is standard deviation of the values of attribute i for class non j.

The proposed method in study is Absolute Correlation Weighted Naïve Bayes (AC-WNB) that can be described in Figure 1. There are three different processes in AC-WNB model figured with shaded block, which are calculate the weight, calculate the weighted likelihood and calculate prior. Calculate the weight is a process that calculates weight for each attribute in the training process. The weight is calculated the absolute value of correlation coefficient by using the mean and standard deviation of each attribute. Calculate the weighted likelihood is a process that calculates the likelihood for each attribute to classify the testing dataset. These likelihood squares by weight that generated on calculating the weight process. Calculate the prior used weighted likelihood. The weighted likelihood of class was divided by sum of all weighted likelihood.

As shown on Figure 1, dataset is divided into data training and data testing using 10-fold cross validation method. Data training is used to training process. In training process, means, standard deviations and weight of each attributes will be calculated. Weight will be calculated using mean and standard deviation. The absolute value of the difference between mean of class and the other classes that have been divided with the summation of standard deviations is used as weight value.

The detail of training process of AC-WNB as follows:

1. Calculate the mean value of attribute in each class
   The mean value obtained by summing all instances value on a specific attribute then divide by the number of instances. The mean value can be formulated as follows
   $$\mu = \frac{x_1 + x_2 + \cdots + x_n}{n}$$
2. Calculate the standard deviation of attribute in each class
   The standard deviation is the square root value of the variance. The variance is obtained by subtract each value with mean value and square the result then divided by the number of instances. There are two types of standard deviation that are the sample and population. The population divide the square results with the number of instances (N) and the sample divide the square results with the number of instances minus one (N-1). Absolute Correlation based Weighted Naïve Bayes use the sample to calculate standard deviation. The standard deviation can be formulated as follows
   $$\sigma = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2$$
3. Calculate the weight of attribute
   The weight can be obtained by calculation using mean and standard deviation that formulated as follows
   $$w_i = \left|\frac{(\mu_{ij} - \mu_{i\bar{j}})}{(\sigma_{ij} + \sigma_{i\bar{j}})}\right|$$

4. Repeat steps 1-3 for all attribute

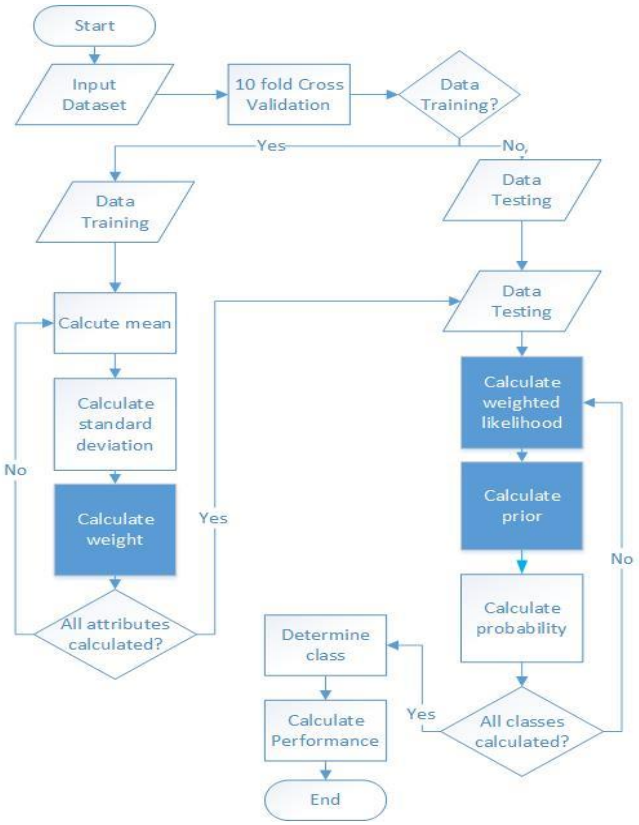


Figure 1 Flow Chart of AC-WNB Method

The model that has been generated in training process will be tested using data testing. AC-WNB has to calculate likelihood, prior, weighted likelihood and weighted prior to classifying the class. Likelihood is commonly known as conditional probabilities that is calculated using normal distribution of each attribute. This normal distribution of each attribute would be squared by weight that has been generated at training process. The product (π) of weighted normal distribution of all attributes is used to weighted likelihood value of class. The prior is also known as class probabilities. The value of prior of class calculated by divided weighted likelihood of class with sum of all weighted likelihood class. Then, weighted likelihood value would be multiplied with prior to classify. The detail of testing process of AC-WNB as follows:

1. Calculate weighted likelihood
   For numeric attribute, likelihood calculates by using normal distribution. In this study, all attributes were numeric. The normal distribution was formulated as follows

$$p(a|c) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

   Then, weighted likelihood can be calculated with formula as follows

$$L = \prod_{i=1}^{n} p(a_i|c)^{w_i}$$

2. Calculate prior
   The prior of class calculated by divided weighted likelihood of class with sum of all weighted likelihood class. The prior can be formulated as follows

$$p(c) = \frac{\prod_{i=1}^{n} p(a_i|c)^{w_i}}{\prod_{i=1}^{n} p(a_i|c)^{w_i} + \prod_{i=1}^{n} p(a_i|\bar{c})^{w_i}}$$

3. Calculate probabilities of class
   In order to calculate probability of class, weighted likelihood value will be multiplied prior. The probabilities of class can be formulated as follows

$$p(c|E) = p(c) \prod_{i=1}^{n} p(a_i|c)$$

4. Repeat steps1-3 for all class
5. The predicted class was determined by the highest probability.

4 DATA GATHERING

NASA dataset that was used in this study was obtained from the MDP (Metric Data Program) Repository ("NASA-SoftwareDefectDataSets," n.d.), which can also be obtained from the PROMISE Repository. NASA datasets have been widely used for research in the field of software engineering(T. Hall et al., 2012). This dataset is devoted to research on the topic of software defect and software failures. Therefore, most studies use this dataset to do some research, ranging from doing predictions, associations, and to the development of a model for research.

NASA dataset currently available from many Repository (MDP and PROMISE), therefore, the researchers used a dataset derived from the MDP Repository who has been repaired by Martin Shepherd(Shepperd, Song, Sun, & Mair, 2013), with the following specifications(Gray, Bowes, Davey, Sun, & Christianson, 2012) in Table 1. All attributes in NASA MDP dataset are numeric. Dataset has been fixed by removing data null or no value. Accordingly, this study used the dataset to do some research in determining the proposed model in the prediction of software defects.

Table 1 Dataset Specifications

Attribute	NASA MDP Dataset									
	CM 1	JM 1	KC 1	KC 3	MC 1	MC 2	PC 1	PC3	PC 4	PC 5
LOC_BLANK	√	√	√	√	√	√	√	√	√	√
BRANCH_COUNT	√	√	√	√	√	√	√	√	√	√
CALL_PAIRS	√			√	√	√	√	√	√	√
LOC_CODE_AND_COMMENT	√	√	√	√	√	√	√	√	√	√
LOC_COMMENTS	√	√	√	√	√	√	√	√	√	√
CONDITION_COUNT	√			√	√	√	√	√	√	√
CYCLOMATIC_COMPLEXITY	√	√	√	√	√	√	√	√	√	√
CYCLOMATIC_DENSITY	√			√	√	√	√	√	√	√
DECISION_COUNT	√			√	√	√	√	√	√	√
DECISION_DENSITY	√			√		√	√	√	√	
DESIGN_COMPLEXITY	√	√	√	√	√	√	√	√	√	√
DESIGN_DENSITY	√			√	√	√	√	√	√	√
EDGE_COUNT	√			√	√	√	√	√	√	√
ESSENTIAL_COMPLEXITY	√	√	√	√	√	√	√	√	√	√
ESSENTIAL_DENSITY	√			√	√	√	√	√	√	√
LOC_EXECUTABLE	√	√	√	√	√	√	√	√	√	√
PARAMETER_COUNT	√			√	√	√	√	√	√	√
GLOBAL_DATA_COMPLEXITY				√	√	√				√
GLOBAL_DATA_DENSITY				√	√	√				√
HALSTEAD_CONTENT	√	√	√	√	√	√	√	√	√	√
HALSTEAD_DIFFICULTY	√	√	√	√	√	√	√	√	√	√
HALSTEAD_EFFORT	√	√	√	√	√	√	√	√	√	√
HALSTEAD_ERROR_EST	√	√	√	√	√	√	√	√	√	√
HALSTEAD_LENGTH	√	√	√	√	√	√	√	√	√	√
HALSTEAD_LEVEL	√	√	√	√	√	√	√	√	√	√
HALSTEAD_PROG_TIME	√	√	√	√	√	√	√	√	√	√
HALSTEAD_VOLUME	√	√	√	√	√	√	√	√	√	√
MAINTENANCE_SEVERITY	√			√	√	√	√	√	√	√
MODIFIED_CONDITION_COUNT	√			√	√	√	√	√	√	√
MULTIPLE_CONDITION_COUNT	√			√	√	√	√	√	√	√
NODE_COUNT	√			√	√	√	√	√	√	√
NORMALIZED_CYLOMATIC_COMPLEXITY	√			√	√	√	√	√	√	√
NUM_OPERANDS	√	√	√	√	√	√	√	√	√	√
NUM_OPERATORS	√	√	√	√	√	√	√	√	√	√
NUM_UNIQUE_OPERANDS	√	√	√	√	√	√	√	√	√	√
NUM_UNIQUE_OPERATORS	√	√	√	√	√	√	√	√	√	√
NUMBER_OF_LINES	√			√	√	√	√	√	√	√
PERCENT_COMMENTS	√			√	√	√	√	√	√	√
LOC_TOTAL	√	√	√	√	√	√	√	√	√	√
Number of Code Attributes	37	21	21	39	38	39	37	37	37	38
Programming Language	C	C	C++	Java	C and C++	C	C	C	C	C++
Number of Modules	327	7782	1183	194	1988	125	705	1077	1287	1711
Number of defect Modules	42	1672	314	36	46	44	61	134	177	471

5 RESULT AND ANALYSIS

The proposed model was developed using Java using NetBeans IDE 7.3.1. Datasets that used in this study are: CM1, JM1, KC1, KC3, MC1, MC2, PC1, PC3, PC4 and PC5. The weight values that calculated by AC-WNB shown in Table 2.

Table 2 Weight of attribute

Attribute	NASA MDP Dataset									
	CM 1	JM 1	KC 1	KC 3	MC 1	MC 2	PC 1	PC 3	PC 4	PC 5
LOC_BLANK	0.18	0.21	0.25	0.34	0.37	0.37	0.38	0.49	0.24	0.15
BRANCH_COUNT	0.19	0.20	0.22	0.25	0.20	0.37	0.19	0.09	0.01	0.23
CALL_PAIRS	0.28			0.30	0.29	0.24	0.24	0.24	0.11	0.21
LOC_CODE_AND_COMMENT	0.04	0.12	0.05	0.38	0.42	0.17	0.32	0.29	0.48	0.28
LOC_COMMENTS	0.35	0.17	0.17	0.10	0.31	0.39	0.44	0.38	0.11	0.19
CONDITION_COUNT	0.19			0.20	0.17	0.38	0.18	0.08	0.22	0.21
CYCLOMATIC_COMPLEXITY	0.19	0.18	0.22	0.26	0.12	0.37	0.20	0.09	0.01	0.23
CYCLOMATIC_DENSITY	0.28			0.19	0.29	0.07	0.52	0.21	0.33	0.22
DECISION_COUNT	0.19			0.20	0.16	0.38	0.17	0.07	0.23	0.20
DECISION_DENSITY	0.03			0.00		0.16	0.01	0.13	0.48	
DESIGN_COMPLEXITY	0.23	0.16	0.22	0.26	0.03	0.31	0.19	0.09	0.06	0.23
DESIGN_DENSITY	0.16			0.03	0.12	0.13	0.05	0.00	0.13	0.06
EDGE_COUNT	0.20			0.28	0.22	0.40	0.20	0.10	0.05	0.22
ESSENTIAL_COMPLEXITY	0.12	0.14	0.15	0.17	0.13	0.36	0.13	0.01	0.14	0.18
ESSENTIAL_DENSITY	0.00			0.05	0.00	0.34	0.01	0.03	0.11	0.11
LOC_EXECUTABLE	0.29	0.21	0.25	0.29	0.24	0.34	0.33	0.13	0.19	0.19
PARAMETER_COUNT	0.07			0.06	0.00	0.14	0.17	0.15	0.14	0.07
GLOBAL_DATA_COMPLEXITY				0.26	0.01	0.34				0.21
GLOBAL_DATA_DENSITY				0.11	0.23	0.06				0.09
HALSTEAD_CONTENT	0.31	0.22	0.24	0.31	0.23	0.11	0.47	0.20	0.11	0.03
HALSTEAD_DIFFICULTY	0.21	0.18	0.30	0.22	0.13	0.11	0.15	0.04	0.15	0.28
HALSTEAD_EFFORT	0.14	0.09	0.22	0.23	0.04	0.38	0.16	0.02	0.13	0.18
HALSTEAD_ERROR_EST	0.26	0.20	0.27	0.26	0.20	0.34	0.29	0.08	0.17	0.13
HALSTEAD_LENGTH	0.27	0.22	0.28	0.27	0.21	0.35	0.31	0.11	0.19	0.16
HALSTEAD_LEVEL	0.27	0.14	0.18	0.21	0.09	0.34	0.24	0.21	0.18	0.30
HALSTEAD_PROG_TIME	0.14	0.09	0.22	0.23	0.04	0.38	0.16	0.02	0.13	0.18
HALSTEAD_VOLUME	0.26	0.20	0.27	0.26	0.20	0.34	0.29	0.08	0.17	0.13
MAINTENANCE_SEVERITY	0.13			0.09	0.01	0.18	0.14	0.16	0.28	0.21
MODIFIED_CONDITION_COUNT	0.19			0.20	0.18	0.37	0.18	0.09	0.22	0.22
MULTIPLE_CONDITION_COUNT	0.19			0.20	0.17	0.38	0.18	0.09	0.22	0.20
NODE_COUNT	0.19			0.28	0.22	0.40	0.19	0.10	0.06	0.21
NORMALIZED_CYLOMATIC_COMPLEXITY	0.32			0.28	0.26	0.13	0.54	0.27	0.37	0.14
NUM_OPERANDS	0.26	0.22	0.28	0.26	0.20	0.35	0.31	0.11	0.18	0.17
NUM_OPERATORS	0.27	0.21	0.27	0.28	0.21	0.34	0.31	0.11	0.18	0.15
NUM_UNIQUE_OPERANDS	0.31	0.23	0.30	0.29	0.31	0.21	0.37	0.24	0.15	0.14
NUM_UNIQUE_OPERATORS	0.34	0.17	0.30	0.26	0.24	0.41	0.33	0.19	0.14	0.34
NUMBER_OF_LINES	0.30			0.30	0.33	0.37	0.43	0.29	0.23	0.20
PERCENT_COMMENTS	0.29			0.22	0.47	0.22	0.38	0.43	0.44	0.10
LOC_TOTAL	0.26	0.22	0.26	0.29	0.28	0.34	0.35	0.16	0.44	0.20

One of the performance indicators to evaluate the performance of the classifier in the experiment, area under the curve (AUC) was applied. AUC used in this study to evaluated the classifier on class imbalance data as recommended by Lessmann et al. (Lessmann, Baesens, Mues, & Pietsch, 2008) and Brown et al.(Brown & Mues, 2012). The results of all the methods would be compared using Wilcoxon signed-rank test to verify whether there is a significant difference between methods.

Table 3 Performance of the NB and AC-WNB Model

Dataset	NB		AC-WNB		AC-WACNB	
	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC
CM1	81.04	0.768	81.04	0.814	81.35	0.857
JM1	78.09	0.803	78.17	0.810	78.08	0.813
KC1	72.02	0.793	72.19	0.807	72.19	0.834
KC3	79.38	0.830	79.38	0.879	79.38	0.851
MC1	88.58	0.836	88.98	0.855	89.03	0.833
MC2	72.00	0.840	72.80	0.843	72.00	0.817
PC1	87.66	0.800	87.94	0.817	87.80	0.847
PC3	28.04	0.873	69.73	0.862	70.66	0.890
PC4	86.01	0.797	82.28	0.838	82.13	0.906
PC5	74.63	0.782	74.93	0.805	74.93	0.852

As shown in Table 3, Absolute Correlation based Weighted Naïve Bayes have better average accuracy that is 78.74%, followed by Naïve Bayes with 74.74%, while for AUC values, the average of AUC values of Absolute Correlation Weighted Naïve Bayes is higher than Naïve Bayes. The average of AUC values of Absolute Correlation Weighted Naïve Bayes is 0.833, and then Naïve Bayes is 0.812

The results of the comparison of the AUC can be described in Figure 2. The AUC values of AC-WNB higher than NB. It can be concluded that Absolute Correlation based Weighted Attribute-class Naïve Bayes is much better than others method. However, this increase should be examined more deeply with significance test.

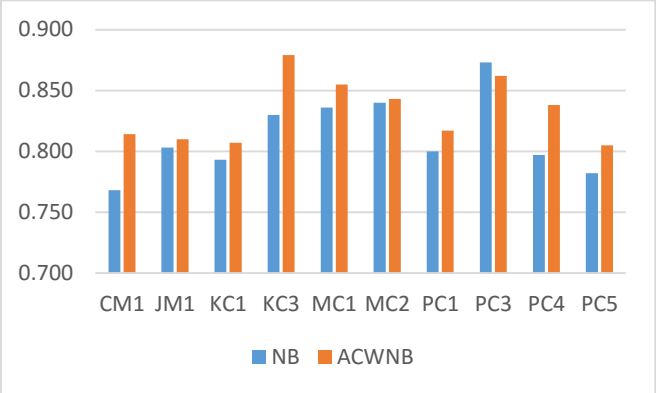


Figure 2 Performance (AUC) of the Models

AUC values of Naïve Bayes and Absolute Correlation based Weighted Naïve Bayes would be compared using Wilcoxon signed-rank test. A significant different in performance was considered when the results of Wilcoxon signed-rank test showed that P-value<alpha (0.05). Wilcoxon signed-rank test results on the statistical test of AUC of Naïve Bayes and Absolute Correlation based Weighted Naïve Bayes was shown in Table 4. The average of AUC values for AC-WNB was higher than NB that is 0.833 with P-value 0.01. As the computed P-value was lower than the significance level alpha, the null hypothesis (H0) that is the two samples follow the same distribution should be rejected and accept the alternative hypothesis (Ha) that is the distributions of the two samples are different. It means that NB and AC-WNB had significant differences P-value<alpha (0.05). Therefore, it can be concluded that AC-WNB makes an improvement when compared with NB in prediction performance. Hence, it means that the absolute value of correlation coefficient can improve the performance of Naïve Bayes for classifying on software defect prediction

Table 4 Wilcoxon Signed-Rank Test of AUC of NB and AC-WNB

	NB	*AC-WNB*
Observation	10	10
Mean	0.8122	0.833
Median	0.8015	0.8275
Standard Deviation	0.030085877	0.02484351
The Test Procedure		
Hypothetical Mean Difference	0	
Nb. Of Zero Differences	0	
Rank Sum	55	
Rank Average	5.5	
Test Statistic (S+)	52	
Significance Level	0.05	
Exact Procedure		
Critical Value	8	
Decision Rule	Reject H0 if (S+) > 8	
Final Decision	The Null Hypothesis Cannot be Rejected due to Insufficient Evidence in the Sample	
P-Value	0.01	

6 CONCLUSION

Naive Bayes proved effective in predicting software defects. However, Naïve Bayes perform less well for predicting software defects due to the assumption that all attributes are equally important and are not related to each other while, in fact, this assumption is not true in many cases. One of the ways to develop Naïve Bayes is set a different weight value of each attributes.

Absolute Correlation Weighted Naïve Bayes has been proposed. Absolute Correlation Weighted Naïve Bayes provides weight to each attribute. The weight values are depending on how relevant the attribute with class. The correlation coefficient is used to measure the relevance between class and attribute. Therefore, absolute value of correlation coefficients was used as weight at Absolute Correlation Weighted Naïve Bayes.

The results of experiments showed that the mean of AUC value of Naïve Bayes for classifying software defect was 0.8122, while the average AUC value of Absolute Correlation Weighted Naïve Bayes was 0.833. The results of Wilcoxon signed-rank test showed that Absolute Correlation Weighted Naïve Bayes had significant differences with Naïve Bayes P-value 0.008 < alpha (0.05). Therefore, it can be concluded that absolute correlation coefficient can improve the performance of Naïve Bayes for classifying on software defect prediction.

7 FUTURE WORK

In this study, absolute coefficient correlation could improve the performance of Naive Bayes for classifying on software defect prediction. Absolute correlation was used as weight for attribute by calculated the relevance between attribute. It used to measure the strength of relationship between two attributes(Freund & Wilson, 2003). Taheri et al.(Taheri et al., 2013) proposed a new attribute weighted Naïve Bayes classifier, called AWNB, which provide more than one weight for every attribute. Therefore, investigation to improve the performance of Naive Bayes for classifying on software defect prediction by using absolute correlation to provide more than one weight is one of main direction for future work.

REFERENCES

Arauzo-Azofra, A., Aznarte, J. L., & Benítez, J. M. (2011). Empirical study of feature selection methods based on individual feature evaluation for classification problems. *Expert Systems with Applications*, *38*(7), 8170–8177. doi:10.1016/j.eswa.2010.12.160

Arisholm, E., Briand, L. C., & Fuglerud, M. (2007). Data Mining Techniques for Building Fault-proneness Models in Telecom Java Software. In *The 18th IEEE International Symposium on Software Reliability (ISSRE '07)* (pp. 215–224). IEEE. doi:10.1109/ISSRE.2007.22

Arisholm, E., Briand, L. C., & Johannessen, E. B. (2010). A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, *83*(1), 2–17. doi:10.1016/j.jss.2009.06.055

Bibi, S., Tsoumakas, G., Stamelos, I., & Vlahvas, I. (2006). Software Defect Prediction Using Regression via Classification. In *IEEE International Conference on Computer Systems and Applications, 2006.* (pp. 330–336). IEEE. doi:10.1109/AICCSA.2006.205110

Brown, I., & Mues, C. (2012). An experimental comparison of classification algorithms for imbalanced credit scoring data sets. *Expert Systems with Applications*, *39*(3), 3446–3453. doi:10.1016/j.eswa.2011.09.033

Freund, R. J., & Wilson, W. J. (2003). *Statistical Methods* (2nd ed.). Academic Press.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, *29*(2-3), 131–163.

Furey, T. S., Cristianini, N., Duffy, N., Bednarski, D. W., Schummer, M., & Haussler, D. (2000). Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, *16*(10), 906–914. doi:10.1093/bioinformatics/16.10.906

Golub, T. R. (1999). Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring. *Science*, *286*(5439), 531–537. doi:10.1126/science.286.5439.531

Gray, D., Bowes, D., Davey, N., Sun, Y., & Christianson, B. (2012). Reflections on the NASA MDP data sets. *IET Software*, *6*(6), 549. doi:10.1049/iet-sen.2011.0132

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, *46*(1-3), 389–422. doi:10.1023/A:1012487302797

Hall, M. (2007). A decision tree-based attribute weighting filter for naive Bayes. *Knowledge-Based Systems*, *20*(2), 120–126. doi:10.1016/j.knosys.2006.11.008

Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Transactions on Software Engineering*, *38*(6), 1276–1304. doi:10.1109/TSE.2011.103

Harrington, P. (2012). *Machine Learning in Action*. Connecticut: Manning Publications Co. Greenwich, CT, USA.

Hsu, C., Chang, C., & Lin, C. (2003). A Practical Guide to Support Vector Classification. *Department of Computer Science and Information Engineering, National Taiwai University*.

Jiang, L. (2011). Random one-dependence estimators. *Pattern Recognition Letters*, *32*(3), 532–539. doi:10.1016/j.patrec.2010.11.016

Jiang, L., Cai, Z., & Wang, D. (2010). Improving Naive Bayes for Classification. *International Journal of Computers and Applications*, *32*(3). doi:10.2316/Journal.202.2010.3.202-2747

Jiang, L., Wang, D., Cai, Z., & Yan, X. (2007). Survey of improving naive Bayes for classification. *Advanced Data Mining and Applications*. doi:10.1007/978-3-540-73871-8_14

Khoshgoftaar, T. M., & Seliya, N. (2004). Comparative Assessment of Software Quality Classification Techniques: An Empirical Case Study. *Empirical Software Engineering*, *9*(3), 229–257. doi:10.1023/B:EMSE.0000027781.18360.9b

Khoshgoftaar, T. M., Yuan, X., Allen, E. B., Jones, W. D., & Hudepohl, J. P. (2002). Uncertain Classification of Fault-Prone Software Modules. *Empirical Software Engineering*, *7*(4), 297–318. doi:10.1023/A:1020511004267

Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *IEEE Transactions on Software Engineering*, *34*(4), 485–496. doi:10.1109/TSE.2008.35

Lin, J., & Yu, J. (2011). Weighted Naive Bayes classification algorithm based on particle swarm optimization. In *2011 IEEE 3rd International Conference on Communication Software and Networks* (pp. 444–447). IEEE. doi:10.1109/ICCSN.2011.6014307

Mizuno, O., & Kikuno, T. (2007). Training on errors experiment to detect fault-prone software modules by spam filter. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering - ESEC-FSE '07* (p. 405). New York, New York, USA: ACM Press. doi:10.1145/1287624.1287683

NASA-SoftwareDefectDataSets. (n.d.). Retrieved from http://nasa-softwaredefectdatasets.wikispaces.com/

Okutan, A., & Yıldız, O. T. (2012). Software defect prediction using Bayesian networks. *Empirical Software Engineering*, *19*(1), 154–181. doi:10.1007/s10664-012-9218-8

Pavlidis, P., Weston, J., Cai, J., & Grundy, W. N. (2001). Gene functional classification from heterogeneous data. In

Proceedings of the fifth annual international conference on Computational biology - RECOMB '01 (pp. 249–255). New York, New York, USA: ACM Press. doi:10.1145/369133.369228

Ratanamahatana, C., & Gunopulos, D. (2003). Feature selection for the naive bayesian classifier using decision trees. *Applied Artificial Intelligence*, 475–487. doi:10.1080/08839510390219327

Shepperd, M., Song, Q., Sun, Z., & Mair, C. (2013). Data Quality: Some Comments on the NASA Software Defect Datasets. *IEEE Transactions on Software Engineering*, *39*(9), 1208–1215. doi:10.1109/TSE.2013.11

Song, Q., Jia, Z., Shepperd, M., Ying, S., & Liu, J. (2011). A General Software Defect-Proneness Prediction Framework. *IEEE Transactions on Software Engineering*, *37*(3), 356–370. doi:10.1109/TSE.2010.90

Taheri, S., Yearwood, J., Mammadov, M., & Seifollahi, S. (2013). Attribute weighted Naive Bayes classifier using a local optimization. *Neural Computing and Applications*. doi:10.1007/s00521-012-1329-z

Turhan, B., & Bener, A. (2009). Analysis of Naive Bayes' assumptions on software fault data: An empirical study. *Data & Knowledge Engineering*, *68*(2), 278–290. doi:10.1016/j.datak.2008.10.005

Webb, G. I., Boughton, J. R., & Wang, Z. (2005). Not So Naive Bayes: Aggregating One-Dependence Estimators. *Machine Learning*, *58*(1), 5–24. doi:10.1007/s10994-005-4258-6

Webb, G. I., Boughton, J. R., Zheng, F., Ting, K. M., & Salem, H. (2011). Learning by extrapolation from marginal to full-multivariate probability distributions: decreasingly naive Bayesian classification. *Machine Learning*, *86*(2), 233–272. doi:10.1007/s10994-011-5263-6

Wu, J., & Cai, Z. (2011). Attribute weighting via differential evolution algorithm for attribute weighted naive bayes (WNB). *Journal of Computational Information Systems*, *7*(12), 1672–1679.

Wu, X., & Kumar, V. (2009). *The top ten algorithms in data mining*. *International Statistical Review* (Vol. 78, pp. 158–158). Taylor & Francis Group.

Zaidi, N., Cerquides, J., Carman, M., & Webb, G. (2013). Alleviating Naive Bayes Attribute Independence Assumption by Attribute Weighting. *Journal of Machine Learning Research*, *14*, 1947–1988.

Zhang, H. (2004). Learning Weighted Naive Bayes with Accurate Ranking. In *Fourth IEEE International Conference on Data Mining (ICDM'04)* (pp. 567–570). IEEE. doi:10.1109/ICDM.2004.10030

Arauzo-Azofra, A., Aznarte, J. L., & Benítez, J. M. (2011). Empirical study of feature selection methods based on individual feature evaluation for classification problems. *Expert Systems with Applications*, *38*(7), 8170–8177. doi:10.1016/j.eswa.2010.12.160

Arisholm, E., Briand, L. C., & Fuglerud, M. (2007). Data Mining Techniques for Building Fault-proneness Models in Telecom Java Software. In *The 18th IEEE International Symposium on Software Reliability (ISSRE '07)* (pp. 215–224). IEEE. doi:10.1109/ISSRE.2007.22

Arisholm, E., Briand, L. C., & Johannessen, E. B. (2010). A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, *83*(1), 2–17. doi:10.1016/j.jss.2009.06.055

Bibi, S., Tsoumakas, G., Stamelos, I., & Vlahvas, I. (2006). Software Defect Prediction Using Regression via Classification. In *IEEE International Conference on Computer Systems and Applications, 2006.* (pp. 330–336). IEEE. doi:10.1109/AICCSA.2006.205110

Brown, I., & Mues, C. (2012). An experimental comparison of classification algorithms for imbalanced credit scoring data sets. *Expert Systems with Applications*, *39*(3), 3446–3453. doi:10.1016/j.eswa.2011.09.033

Freund, R. J., & Wilson, W. J. (2003). *Statistical Methods* (2nd ed.). Academic Press.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, *29*(2-3), 131–163.

Furey, T. S., Cristianini, N., Duffy, N., Bednarski, D. W., Schummer, M., & Haussler, D. (2000). Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, *16*(10), 906–914. doi:10.1093/bioinformatics/16.10.906

Golub, T. R. (1999). Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring. *Science*, *286*(5439), 531–537. doi:10.1126/science.286.5439.531

Gray, D., Bowes, D., Davey, N., Sun, Y., & Christianson, B. (2012). Reflections on the NASA MDP data sets. *IET Software*, *6*(6), 549. doi:10.1049/iet-sen.2011.0132

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, *46*(1-3), 389–422. doi:10.1023/A:1012487302797

Hall, M. (2007). A decision tree-based attribute weighting filter for naive Bayes. *Knowledge-Based Systems*, *20*(2), 120–126. doi:10.1016/j.knosys.2006.11.008

Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Transactions on Software Engineering*, *38*(6), 1276–1304. doi:10.1109/TSE.2011.103

Harrington, P. (2012). *Machine Learning in Action*. Connecticut: Manning Publications Co. Greenwich, CT, USA.

Hsu, C., Chang, C., & Lin, C. (2003). A Practical Guide to Support Vector Classification. *Department of Computer Science and Information Engineering, National Taiwai University*.

Jiang, L. (2011). Random one-dependence estimators. *Pattern Recognition Letters*, *32*(3), 532–539. doi:10.1016/j.patrec.2010.11.016

Jiang, L., Cai, Z., & Wang, D. (2010). Improving Naive Bayes for Classification. *International Journal of Computers and Applications*, *32*(3). doi:10.2316/Journal.202.2010.3.202-2747

Jiang, L., Wang, D., Cai, Z., & Yan, X. (2007). Survey of improving naive Bayes for classification. *Advanced Data Mining and Applications*. doi:10.1007/978-3-540-73871-8_14

Khoshgoftaar, T. M., & Seliya, N. (2004). Comparative Assessment of Software Quality Classification Techniques: An Empirical Case Study. *Empirical Software Engineering*, *9*(3), 229–257. doi:10.1023/B:EMSE.0000027781.18360.9b

Khoshgoftaar, T. M., Yuan, X., Allen, E. B., Jones, W. D., & Hudepohl, J. P. (2002). Uncertain Classification of Fault-Prone Software Modules. *Empirical Software Engineering*, *7*(4), 297–318. doi:10.1023/A:1020511004267

Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *IEEE Transactions on Software Engineering*, *34*(4), 485–496. doi:10.1109/TSE.2008.35

Lin, J., & Yu, J. (2011). Weighted Naive Bayes classification algorithm based on particle swarm optimization. In *2011 IEEE 3rd International Conference on Communication Software and Networks* (pp. 444–447). IEEE. doi:10.1109/ICCSN.2011.6014307

Mizuno, O., & Kikuno, T. (2007). Training on errors experiment to detect fault-prone software modules by spam filter. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering - ESEC-FSE '07* (p. 405). New York, New York, USA: ACM Press. doi:10.1145/1287624.1287683

NASA-SoftwareDefectDataSets. (n.d.). Retrieved from http://nasa-softwaredefectdatasets.wikispaces.com/

Okutan, A., & Yıldız, O. T. (2012). Software defect prediction using Bayesian networks. *Empirical Software Engineering*, *19*(1), 154–181. doi:10.1007/s10664-012-9218-8

Pavlidis, P., Weston, J., Cai, J., & Grundy, W. N. (2001). Gene functional classification from heterogeneous data. In *Proceedings of the fifth annual international conference on Computational biology - RECOMB '01* (pp. 249–255). New York, New York, USA: ACM Press. doi:10.1145/369133.369228

Ratanamahatana, C., & Gunopulos, D. (2003). Feature selection for the naive bayesian classifier using decision trees. *Applied Artificial Intelligence*, 475–487. doi:10.1080/08839510390219327

Shepperd, M., Song, Q., Sun, Z., & Mair, C. (2013). Data Quality: Some Comments on the NASA Software Defect Datasets. *IEEE Transactions on Software Engineering*, *39*(9), 1208–1215. doi:10.1109/TSE.2013.11

Song, Q., Jia, Z., Shepperd, M., Ying, S., & Liu, J. (2011). A General Software Defect-Proneness Prediction Framework. *IEEE Transactions on Software Engineering*, *37*(3), 356–370. doi:10.1109/TSE.2010.90

Taheri, S., Yearwood, J., Mammadov, M., & Seifollahi, S. (2013). Attribute weighted Naive Bayes classifier using a local optimization. *Neural Computing and Applications*. doi:10.1007/s00521-012-1329-z

Turhan, B., & Bener, A. (2009). Analysis of Naive Bayes' assumptions on software fault data: An empirical study. *Data & Knowledge Engineering*, *68*(2), 278–290. doi:10.1016/j.datak.2008.10.005

Webb, G. I., Boughton, J. R., & Wang, Z. (2005). Not So Naive Bayes: Aggregating One-Dependence Estimators. *Machine Learning*, *58*(1), 5–24. doi:10.1007/s10994-005-4258-6

Webb, G. I., Boughton, J. R., Zheng, F., Ting, K. M., & Salem, H. (2011). Learning by extrapolation from marginal to full-multivariate probability distributions: decreasingly naive Bayesian classification. *Machine Learning*, *86*(2), 233–272. doi:10.1007/s10994-011-5263-6

Wu, J., & Cai, Z. (2011). Attribute weighting via differential evolution algorithm for attribute weighted naive bayes (WNB). *Journal of Computational Information Systems*, *7*(12), 1672–1679.

Wu, X., & Kumar, V. (2009). *The top ten algorithms in data mining*. *International Statistical Review* (Vol. 78, pp. 158–158). Taylor & Francis Group.

Zaidi, N., Cerquides, J., Carman, M., & Webb, G. (2013). Alleviating Naive Bayes Attribute Independence Assumption by Attribute Weighting. *Journal of Machine Learning Research*, *14*, 1947–1988.

Zhang, H. (2004). Learning Weighted Naive Bayes with Accurate Ranking. In *Fourth IEEE International Conference on Data Mining (ICDM'04)* (pp. 567–570). IEEE. doi:10.1109/ICDM.2004.10030

BIOGRAPHY OF AUTHORS

Rizky Tri Asmono Received bachelor degree and master degree in Computer Science in 2012 and 2014 from Dian Nuswantoro University, Indonesia. He is an IT Executive at PT. Sree International Indonesia in Indonesia. His current research interests are Software Engineering, Data Mining and Machine Learning.

Romi Satria Wahono. Received B.Eng and M.Eng degrees in Computer Science respectively from Saitama University, Japan, and Ph.D in Software Engineering from Universiti Teknikal Malaysia Melaka. He is a lecturer at the Faculty of Computer Science, Dian Nuswantoro University, Semarang, Indonesia. He is also a founder and chief executive officer of PT Brainmatics Cipta Informatika, a software development company in Indonesia. His current research interests include software engineering and machine learning. Professional member of the ACM, PMI and IEEE Computer Society

Abdul Syukur. Received bachelor degree in Mathematics from Universitas Diponegoro Semarang, master degree in management from from Universitas Atma Jaya Yogyakarta, and doctoral degree in economic from Universitas Merdeka Malang. He is a lecturer and a dean at the Faculty of Computer Science, Dian Nuswantoro University, Semarang, Indonesia. His current research interests include decision support systems and information management systems.

Resampling Logistic Regression untuk Penanganan Ketidakseimbangan *Class* pada Prediksi Cacat *Software*

Harsih Rianto
Sekolah Tinggi Manajemen Informatika dan Komputer Nusa Mandiri
hrsanto@gmail.com

Romi Satria Wahono
Fakultas Ilmu Komputer, Uni versitas Dian Nuswantoro
romi@romisatriawahono.net

Abstract: *Software* yang berkualitas tinggi adalah *software* yang dapat membantu proses bisnis perusahaan dengan efektif, efesien dan tidak ditemukan cacat selama proses pengujian, pemeriksaan, dan implementasi. Perbaikan *software* setelah pengirimana dan implementasi, membutuhkan biaya jauh lebih mahal dari pada saat pengembangan. Biaya yang dibutuhkan untuk pengujian *software* menghabisakan lebih dari 50% dari biaya pengembangan. Dibutuhkan model pengujian cacat *software* untuk mengurangi biaya yang dikeluarkan. Saat ini belum ada model prediksi cacat *software* yang berlaku umum pada saat digunakan digunakan. Model Logistic Regression merupakan model paling efektif dan efesien dalam prediksi cacat *software*. Kelemahan dari Logistic Regression adalah rentan terhadap *underfitting* pada dataset yang kelasnya tidak seimbang, sehingga akan menghasilkan akurasi yang rendah. Dataset NASA MDP adalah dataset umum yang digunakan dalam prediksi cacat *software*. Salah satu karakter dari dataset prediksi cacat software, termasuk didalamnya dataset NASA MDP adalah memiliki ketidakseimbangan pada kelas. Untuk menangani masalah ketidakseimbangan kelas pada dataset cacat software pada penelitian ini diusulkan metode *resampling*. Eksperimen dilakukan untuk membandingkan hasil kinerja Logistic Regression sebelum dan setelah diterapkan metode *resampling*. Demikian juga dilakukan eksperimen untuk membandingkan metode yang diusulkan hasil pengklasifikasi lain seperti Naïve Bayes, Linear Descriminant Analysis, C4.5, Random Forest, Neural Network, k-Nearest Network. Hasil eksperimen menunjukkan bahwa tingkat akurasi Logistic Regression dengan *resampling* lebih tinggi dibandingkan dengan metode Logistric Regression yang tidak menggunakan *resampling*, demikian juga bila dibandingkan dengan pengkalisifkasi yang lain. Dari hasil eksperimen di atas dapat disimpulkan bahwa metode *resampling* terbukti efektif dalam menyelesaikan ketidakseimbangan kelas pada prediksi cacat *software* dengan algoritma Logistic Regression.

Keywords: Ketidakseimbangan Kelas, Logistic Regression, *Resampling*.

1 PENDAHULUAN

Software yang dikembangkan dan dibuat oleh para pengembang sebagian besar digunakan oleh perusahaan atau instansi pemerintahan. Pembuatan *software* tersebut bertujuan agar proses bisnis pada perusahaan atau instansi pemerintahan berjalan dengan efesien, efektif, cepat dan akurat. Pengembangan sebuah *software* yang berkualitas tinggi membutuhkan biaya yang sangat mahal (Chang, Mu, & Zhang, 2011; Czibula, Marian, & Czibula, 2014; Ma, Luo, Zeng, & Chen, 2012; Wahono, Suryana, & Ahmad, 2014). Untuk meningkatkan efesiensi dan jaminan kualitas yang tinggi dari sebuah s*oftware*, dalam pengujiannya diperlukan program yang mampu memprediksi cacat s*oftware* (Chang et al., 2011). Prediksi cacat *Software* digunakan untuk mengidentifikasi modul yang rawan terhadap cacat pada pengembangan modul yang akan dirilis dan membantu memprediksi kesalahan pada modul tersebut.

Penelitian dalam bidang *Software Engineering* khususnya tentang prediksi cacat *software* telah menjadi topik penelitian yang sangat penting (Hall, Beecham, Bowes, Gray, & Counsell, 2012). Saat ini penelitian prediksi cacat s*oftware* fokus pada 1) estimasi jumlah cacat pada s*oftware*, 2) asosiasi cacat pada *software*, 3) klasifikasi pada cacat s*oftware* terutama pada penentuan cacat dan non-cacat (Song, Jia, Shepperd, Ying, & Liu, 2011). Klasifikasi merupakan pendekatan yang poluler untuk memprediksi cacat s*oftware* (Lessmann, Member, Baesens, Mues, & Pietsch, 2008). Para pengembang dapat menghindari hal-hal yang merugikan pengguna dan tim pengembang dari cacat *software* sedini mungkin dengan menggunakan prediksi cacat *software*.

Algoritma klasifikasi seperti C4.5, Decision Tree, Linear Regression, Logistic Regression (LR), Naïve Bayes (NB), Neural Network (NN), Random Fores (RF) dan Support Vector Machine (SVM) menjadi focus topik penelitian yang banyak dilakukan (Hall et al., 2012). Hasil komparasi algoritma klasifikasi diperoleh dua metode algoritma terbaik yaitu Naïve Bayes dan Logistic Regression (Hall et al., 2012). Naïve Bayes adalah model klasifikasi probabilitas sederhana. Penggunaan algoritma Naïve Bayes sangat mudah dan nyaman karena tidak memerlukan estimasi parameter yang rumit. Sehingga Naïve Bayes bisa digunakan pada dataset yang sangat besar. Selain pada dataset yang besar Naïve Bayes juga menyajikan hasil klasifikasi kepada pengguna dengan sangat mudah tanpa harus memiliki pengetahuan teknologi klasifikasi terlebih dahulu (X. Wu & Kumar, 2010). Namun Naïve Bayes berasumsi pada semua atribut dataset adalah sama penting dan tidak terkait satu sama lain, sedangkan pada kenyataannya sulit dipahami keterkaitan antar atribut (J. Wu & Cai, 2011). Logistic Regression adalah metode klasifikasi statistik probabilitas. Keuntungan Logistic Regression adalah algoritma ini telah dipelajar secara ekstensif (Hosmer et al., 2013; Hosmer & Lemeshow, 2000) disamping pengembangan terbaru tentang penerapan *truncate newton* (Lin et al., 2008). Kelemahan dari Logistic Regression adalah rentan terhadap *underfitting* dan memiliki akurasi yang rendah (Harrington, 2012).

Logistic Regression merupakan klasifikasi linier yang telah terbukti menghasilkan klasifikasi yang powerful dengan statistik probabilitas dan menangani masalah klasifikasi multi kelas (Canu & Smola, 2006; Karsmakers, Pelckmans, &

Suykens, 2007). Masalah besar yang dialami oleh algoritma Logistic Regression adalah ketidakseimbangan kelas (*class imbalance*) pada dataset *berdimensi tinggi* (Lin et al., 2008). Jika dilihat dari dataset yang digunakan untuk prediksi cacat *software*, secara umum menggunakan dataset NASA masih mengalami ketidakseimbangan (*imbalance*) kelas (Khoshgoftaar, Gao, Napolitano, & Wald, 2013). Jumlah data yang rawan cacat (*fault-prone*) lebih sedikit dari pada jumlah data yang tidak rawan cacat (*nonfault-prone*). Membangun model klasifikasi prediksi cacat *software* tanpa melakukan pengolahan data awal, tidak akan menghasilkan prediksi yang efektif, karena jika kelas data awal tidak seimbang (*imbalance*) maka hasil prediksi cenderung menghasilkan kelas mayoritas (Khoshgoftaar et al., 2013). Karena rawan cacat merupakan kelas minoritas dari prediksi cacat *software.* Kinerja model prediksi cacat *software* berkurang secara signifikan, dikarenakan dataset yang digunakan mengandung ketidakseimbangan kelas (*class imbalance*). Secara umum seleksi fitur (*feature selection*) digunakan dalam *machine learning* ketika melibatkan dataset berdimensi tinggi dan atribut yang masih mengandung *noise* (Wahono et al., 2014).

Untuk menangani dataset berdimensi tinggi selain seleksi fitur (*feature selection*), dapat juga menggunakan teknik *resampling*. Dengan meningkatkan kelas minoritas dapat meningkatkan kemampuan algoritma *mechine learning* menjadi lebih baik, karena bisa mengenali sampel kelas minoritas dari sampel mayoritas (Thanathamathee & Lursinsap, 2013). *Resampling* merupakan cara yang paling populer untuk mengatasi masalah ini. Terdapat tiga pendekatan dasar untuk mengatasi masalah ketidakseimbangan kelas, yaitu *oversampling* kelas minoritas, *undersampling* kelas mayoritas atau menggunakan metode *hybrid* yang menggunakan dasar dari kedua metode ini. *Resampling* juga sebagai sarana mengubah distribusai kelas minoritas sehingga tidak kurang terwakili ketika training data pada algorita *mechine learning*. Metode *resampling* sudah terkenal diterapkan untuk memecahakn masalah ketidak seimbangan kelas (*class imbalace*) (Thanathamathee & Lursinsap, 2013).

Oversampling adalah metode yang paling sederhana untuk menangani kelas minoritas dengan melakukan *random* kelas selama proses pengambilan sampel. Proses pengambilan sampel dengan teknik *oversampling* ini adalah dengan menduplikasi kelas positif dan dilakukan penyeimbangkan kelas secara acak (Ganganwar, 2012). Namun, karena metode ini menduplikasi kelas positif yang ada dikelas minoritas, kemungkinan terjadi *overfitting* pasti akan terjadi. *Undersampling* hampir sama dengan teknik *oversampling* dengan menghitung selisih kelas mayoritas dan kelas minoritas. Selanjutnya dilakuakn perulangan sebanyak selisih kelas mayoritas dengan kelas minoritas. Selama proses perulangan dilakukan penghapusan terhadap kelas mayoritas sehingga didapatkan jumlah yang sama dengan kelas minoritas.

Pada penelitian ini yang akan dilakukan adalah penerapan *resampling* untuk penyelesaian ketidakseimbagan kelas (*class imbalance*) pada Logistic Regression untuk prediksi cacat *software*, sehingga dapat menghasilkan kinerja yang baik pada dataset yang seimbang.

2 PENELITIAN TERKAIT

Penelitian tentang penanganan ketidakseimbangan kelas pada Logistic Regression telah banyak dilakukan dan telah dipublikasikan. Untuk melakukan penelitian ini perlu ada kajian terhadap penelitian yang terkait sebelumnya agar dapat mengetahui metode apa saja yang digunakan, data seperti apa yang diproses, dan model seperti yang dihasilkan.

Penelitian yang dilakukan oleh Komarek dan Moore (Komarek & Moore, 2005), melakukan penelitan untuk meningkatkan akurasi prediksi model Logistic Regression dengan mengimplementasikan 3 metode yaitu: 1) *Iteratively re-weighted least squares* (IRLS), 2) *Truncated Regularized* IRLS (TR-IRLS), dan 3) *Generic Likelihood Maximization*. Logistic Regression merupakan algoritma klasifikasi dalam data mining yang memiliki performance tinggi. Dalam beberapa implementasi data mining, Logistic Regression dapat mengunguli algoritma lain seperti Naïve Bayes, Support Vector Mechine (SVM), dan K-Nearest Neighbor (KNN). Dalam penelitian ini menggunakan dataset yang dibagi dalam tiga kategori yaitu: 1) pendeteksian link (citeseer, imdb), 2) dataset penelitian (ds2, ds1, ds1.100, ds1.10) dan 3) klasifikasi teks (modapte.sub). Hasil penelitian menunjukan matrik Area Under Curve (AUC) yaitu, TR-IRLS 0,94 citeseer, 0,98 imdb, 0.72 ds2, 0.94 ds1, 0,91 ds1.100 dan 0,84 ds1.10.

Penelitian yang dilakukan oleh Lin, Weng dan Keerthi (Lin et al., 2008), menunjukan hasil konvergensi yang lebih cepat dari pada metode *quasi Newton*. Metode *Truncated Newton* merupakan metode yang telah diakui mampu menangani dataset berdimensi tinggi seperti penelitian (Komarek & Moore, 2005) namun metode ini tidak sepenuhnya digunakan. Penelitian ini merupakan turunan pengunanan model *Newton*, yang menggunakan dataset berdimensi tinggi yaitu a9a dengan 32561 instance, real-sim dengan 72309 instance, news20 dengan 19996 instance, yahoo-japan dengan 176203 instance, rcv1 dengan 677399 instance dan yahoo-korea dengan 460554 instance. Penerapan metode *Trust Region Newton* (TRON) memperlihatkan hasil komputasi 50% lebih cepat dibadingkan metode *limited memory quasi Newton* (LBFGS) penelitian Liu dan Nocedal 1989 dalam (Lin et al., 2008).

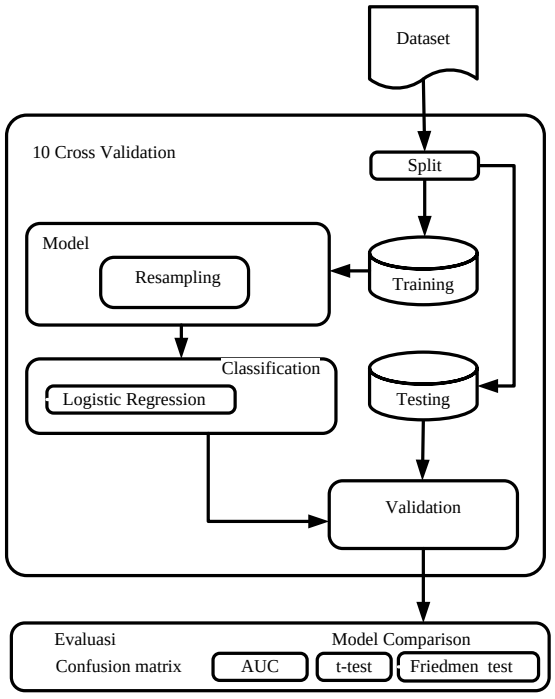
Penelitian yang dilakukan oleh Maalouf dan Trafalis (Maalouf & Trafalis, 2011), mampu menangani dataset yang seimbang dan peristiwa langka. Dalam penelitian ini diterapkan model *rare event re-weighted kernel* logistic regression (RE-WKLR). Dataset yang digunakan dalam penelitian ini adalah UCI Machine Learning dan kejadian nyata angin tornado. Performance dari metode RE-WKLR menunjukan hasil klasifikasi yang lebih tinggi dari pada SVM. Secara keseluruhan hasil akurasi dari penelitian ini dengan menggunakan pengukuran komparasi paired t-test 0.017. Kesimpulan dari penelitian ini menujukan bawah algoritma RE-WKLR sangat mudah diimplementasikan dan kuat dalam menangani data seimbang dan peristiwa langka. Metode RE-WKLR sesuai untuk dataset yang skala kecil dan menengah.

Penelitian yang dilakukan oleh Wahono, Suryana, dan Ahmad (Wahono et al., 2014) menerapan optimasi metaheuristik untuk menemukan solusi optimal dalam seleksi fitur (*feature selection*), secara signifikan mampu mencari solusi berkualitas tinggi dengan jangka waktu yang wajar. Metode yang diusulkan dalam penelitian ini adalah optimasi metaheuristik (algoritma genetika dan *particle swarm optimization* (PSO)) dan teknik Bagging untuk meningkatkan kinerja prediksi cacat *Software*. Bagging baik digunakan untuk model klasifikasi dan regressi. Bagging merupakan algoritma pembelajaran yang stabil pada dataset berdimensi tinggi dan attribut yang masih mengandung *noise* (Alpaydin, 2010). Dalam penelitian ini menggunakan 9 dataset NASA MDP dan 10 algoritma pengklasifikasi dan dikelompokan dalam 5 tipe, yaitu klasifikasi statistic tradisional (Logistic Regression (LR), Linear Discriminant Analysis (LDA), dan Naïve Bayes (NB)), *Nearest Neighbors* (k-Nearest Neighbor (k-NN) dan K*),

Neural Network (Back Propagation (BP), Support Vector Machine (SVM)), dan Decision Tree (C4.5, Classification and Regression Tree (CART), dan Random Forest (RF)). Hasilnya menunjukan bahwa metode yang diusulkan menunjukan peningkatan kenerja model prediksi cacat *Software*. Dari hasil Perbandingan model yang dilakukan, disimpulkan bahwa tidak ada perbedaan signifikan dalam penggunaan optimasi PSO dan algorima genetika saat digunakan pada seleksi fitur, untuk algoritma klasifikasi dalam prediksi cacat *Software*.

3 METODE YANG DIUSULKAN

Metode yang diusulkan pada penelitian ini yaitu untuk meningkatkan kinerja algoritma Logistic Regression dengan metode *resampling* untuk menangani ketidakseimbangan kelas (*class imbalance*) pada prediksi cacat *software*. Selanjutnya untuk validasi menggunakan 10-*fold cross validation.* Hasil pengukuran kinerja algoritma dengan menggunakan uji *t (t-test)* untuk mengetahui perbedaan kinerja model setelah dan sebelum diterapkan model resampling. Selanjutnya juga dilakukan pengujian kinerja model algoritam Logistic Regression dengan algoritma pengklasifikasi lain menggunakan uji Freidmen (*Freidmen test*). Model kerangka pemikiran metode yang diusulkan ditunjukan pada Gambar 1.



Gambar 1. Kerangka Pemikiran Model yang Diusulkan

Dalam penelitian ini dikumpulkan data sekunder yaitu NASA (*National Aeronautics and Space Administration)* MDP (*Metrics Data Program) repository* sebagai *software matrics* yang merupakan dataset yang sudah umum digunakan para peneliti dalam penelitian *Software Engineering* (Hall et al., 2012). Data NASA MDP dikhususkan untuk topik penelitian cacat *software* dan kegagalan *software*. Data NASA tidak hanya terdapat di *repository* MDP namun juga terdapat pada PROMISE. Kebanyakan peneliti menggunakan data NASA dari MDP karena sudah diperbaiki oleh Martin Shepperd (Liebchen & Shepperd, 2008) dengan menghilangkan data yang null atau data yang kosong. Dataset Nasa MDP *Repository* ditunjukan pada Tabel 1.

Tabel 1. Dataset NASA MDP Repository

Nama Atribut	NASA Dataset Repository									
	CM1	JM1	KC1	KC3	MC1	MC2	PC1	PC3	PC4	PC5
LOC BLANK	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LOC CODE AND COMMENT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LOC COMMENTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LOC EXECUTABLE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LOC TOTAL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NUMBER OF LINES	✓			✓	✓	✓	✓	✓	✓	✓
HALSTEAD CONTENT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HALSTEAD DIFFICULTY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HALSTEAD EFFORT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HALSTEAD ERROR EST	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HALSTEAD LENGTH	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HALSTEAD LEVEL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HALSTEAD PROG TIME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HALSTEAD VOLUME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NUM OPERANDS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NUM OPERATORS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NUM UNIQUE OPERANDS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NUM UNIQUE OPERATORS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CYCLOMATIC COMPLEXITY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CYCLOMATIC DENSITY	✓			✓	✓	✓	✓	✓	✓	✓
DESIGN COMPLEXITY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ESSENTIAL COMPLEXITY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
BRANCH COUNT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CALL PAIRS	✓			✓	✓	✓	✓	✓	✓	✓
CONDITION COUNT	✓			✓	✓	✓	✓	✓	✓	✓
DECISION COUNT	✓			✓	✓	✓	✓	✓	✓	✓
DECISION DENSITY	✓			✓		✓	✓	✓	✓	
DESIGN DENSITY	✓			✓	✓	✓	✓	✓	✓	✓
EDGE COUNT	✓			✓	✓	✓	✓	✓	✓	✓
ESSENTIAL DENSITY	✓			✓	✓	✓	✓	✓	✓	✓
GLOBAL DATA COMPLEXITY				✓	✓	✓				
GLOBAL DATA DENSITY				✓	✓	✓				
MAINTENANCE SEVERITY	✓			✓	✓	✓	✓	✓	✓	✓
MODIFIED CONDITION COUNT	✓			✓	✓	✓	✓	✓	✓	✓
MULTIPLE CONDITION COUNT	✓			✓	✓	✓	✓	✓	✓	✓
NODE COUNT	✓			✓	✓	✓	✓	✓	✓	✓
NORMALIZED CYLOMATIC COMPLEXITY	✓			✓	✓	✓	✓	✓	✓	✓
PARAMETER COUNT	✓			✓	✓	✓	✓	✓	✓	✓
PERCENT COMMENTS	✓			✓	✓	✓	✓	✓	✓	✓
PATHOLOGICAL COMPLEXITY										
DEFECTIVE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Jumlah attribute	37	21	21	39	38	39	37	37	37	38
Jumlah modul	344	9593	2096	200	9277	127	759	1125	1399	17001
Jumlah modul cacat	42	1759	325	36	68	44	61	140	178	503
Presentase modul cacat	12%	18%	16%	18%	1%	35%	8%	12%	13%	3%
Jumlah modul tidak cacat	302	7834	1771	164	9209	83	698	985	1221	16498

Proses pengujian metode dimulai dari pembagian dataset dengan metode 10-*fold cross validation* yaitu membagi dataset menjadi dua segmen, segmen pertama digunakan sebagai data training dan segemen kedua digunakan sebagai data testing untuk mevalidasi model (Witten, Frank, & Hall, 2011). Selanjutnya diterapkan tahapan eveluasi menggunakan *Area Under Curve (AUC)* untuk mengukur hasil akurasi indikator dari performa model prediksi. Hasil akurasi dapat dilihat secara manual dengan dilakukan perbandingan klasifikasi menggunakan curva *Receiver Operating Characteristic* (ROC) dari hasil *confusion matrix*. ROC menghasilkan dua garis dengan bentuk *true positives* sebagai garis vertikal dan *false positives* sebagai garis horisontal (Vercellis, 2011). Kurva ROC adalah grafik antara sensitivitas (*true positive rate*), pada sumbu Y dengan 1-spesifitas pada sumbu X (*false positive rate*), curva ROC ini menggambarkan seakan-akan ada terik-menarik antara sumbu Y dengan sumbu X (Dubey, Zhou, Wang, Thompson, & Ye, 2014)

Pengukuran akurasi dengan *confusion matrix* dapat dilihat pada Tabel 2.

Tabel 2. *Confusion Matrix*

Class		Actual	
		TRUE	FALSE
Prediction	TRUE	*True Positive (TP)*	*False Negatif (FN)*
	FALSE	*False Negatif (FN)*	*True Negatif (TN)*

Formulasi perhitungan yang dilakukan (Gorunescu, 2011) adalah sebagai berikut:

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN}$$

$$Sensitivity = TP_{rate} = \frac{TP}{TP + FN}$$

$$Spesificity = TN_{rate} = \frac{TN}{TN + FP}$$

$$FP_{rate} = \frac{FP}{FP + TN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$F\ Measure = \frac{2RP}{R + P}$$

$$G - Mean = \sqrt{sensitivity * specificity}$$

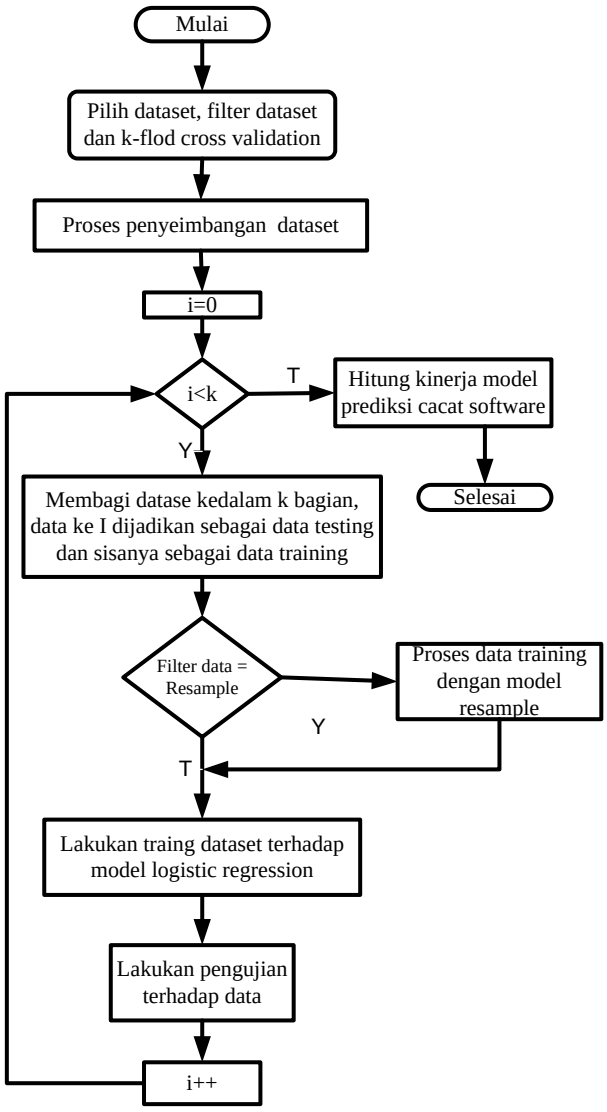
Dalam pengklasifikasi keakuratan dari tes diagnotik menggunakan *Area Under Curve(AUC)* (Gorunescu, 2011) dapat dijabarkan melalui Tabel 3.

Tabel 3. Nilai AUC, Keterangan dan Simbol

Nilai AUC	Klasifikasi	Simbol
0.90 – 1.00	*Excellent classification*	
0.80 – 0.90	*Good classification*	
0.70 – 0.80	*Fair classification*	
0.60 – 0.70	*Poor classification*	
< 0.60	*Failure*	

Evaluasi dalam penelitian ini adalah menggunakan uji *t (t-test).* Uji *t* adalah membandingkan hubungan antara dua variabel yaitu variabel *respon* dan variabel *predictor* (Larose, 2005). Uji *t* sample berpasangan (*paired-sample t-test*) dipergunakan untuk menguji perbandingan selisih dua rata-rata dari dua sample yang berpasangan dengan asumsi bahwa data terdistribusi dengan normal. Selanjutnya untuk mengevaluasi metode Logistic Regression dengan pengklasifikasi lain menggunakan uji Friedman (*Friedman test*). Uji Friedman di usulkan oleh Demsar untuk membandingkan model klasifikasi (Demšar, 2006). Uji Friedman (*Friedman test*) merupakan uji statistik non parametrik, yang juga disebut dengan Anova dua arah berdasarkan peringkat (*two-way anova by ranks*). Uji Friedman (*Friedman test*) berdasarkan peringkat kinerja dari perkiraan kinerja aktual, sehingga lebih tahan terhadap outlier. Semua model klasifikasi akan diperingkat berdasarkan performen terhdap dataset dan peringkat rata-rata dari model klasifikasi yang dibandingkan.

Flowchart metode yang diusulkan dapat dilihat pada Gambar 2, dimulai dengan memilih dataset yang akan diuji, filter dataset dan jumlah *cross validation* yang diinginkan. Selanjutnya dataset diproses kedalam model sebanyak jumlah *cross validation* sampai mendapakan hasil kinerja prediksi cacat *software*. Kemudian dilakukan evaluasi terhadap akurasi, *sensitivity*, *spesificity*, FP_{rate}, *Precision*, *F-Measure*, dan *G-Mean* dari hasil *confusion matrix.* Setelah mendapatkan hasil akurasi dan AUC kemudian dilakukan uji *t* (*t-test*) untuk mengetahui kinerja model setelah dan sebelum diterapakan *resampling*. Untuk mengetahui hasil kinerja model dengan pengklasifikasi lain dilakukan uji Friedman (*Friedman test).*



Gambar 2. Flowchart Metode yang Diusulkan

4 HASIL EKSPERIMEN

Eksperimen yang dilakukan dalam penelitian ini menggunakan sebuah platform komputer berbasis Intel Core i3-3217U @1.80GHz (4 CPUs), RAM 2GB, dan sistem operasi Microsoft Windows 7 Ultimate 32-bit. Sedangkan lingkungan pengembangan aplikasi menggunakan bahasa pemrograman Java Netbeans IDE 8.0.1 dan library Weka 3.6, untuk analisa hasil eksperimen menggunakan aplikasi Microsoft Excel 2007 dengan plugin XLSTAT.

Dalam eksperimen yang dilakukan dengan menggunakan 10 dataset NASA MDP (CM1, JM1, KC1, KC3, MC1, MC2, PC1, PC3, PC4, dan PC5). Metode yang diuji adalah metode pengklasifikasi Logistic Regression (LR), hasil eksperimen disajikan pada Tabel 4, informasi yang disajikan adalah akurasi, *sensitivty* (*recall*), *spesificity*, *Positive Predictive Value* (PPV) atau *Precision*, *Negative predictive Value* (NPV) atau FP_{rate}, *F-Measure*, *G-Mean* dan AUC. Hasil eksperimen pada Tabel 4, menunjukan rata-rata akurasi pada 10 dataset adalah 88.35% dan rata-rata AUC sebesar 0.818.

Tabel 4. Hasil Eksperimen Logistic Regression

Dataset	TP	TN	FP	FN	Accuracy	Recall	Specificity	PPV	NPV	F-Measure	G-Mean	AUC
CM1	9	19	33	283	84.88%	21.43%	93.71%	32.14%	89.56%	0.256	0.448	0.728
JM1	180	137	1579	7697	82.11%	10.23%	98.25%	56.78%	82.98%	0.173	0.317	0.705
KC1	70	44	255	1727	85.74%	21.54%	97.52%	61.40%	87.13%	0.319	0.458	0.800
KC3	12	17	24	147	79.50%	33.33%	89.63%	41.38%	85.97%	0.369	0.547	0.708
MC1	19	14	49	9195	99.32%	27.94%	99.85%	57.58%	99.47%	0.376	0.528	0.875
MC2	36	10	9	72	85.04%	80.00%	87.71%	78.26%	88.89%	0.791	0.038	0.863
PC1	11	13	50	658	91.70%	18.03%	98.14%	45.83%	93.20%	0.259	0.421	0.828
PC3	28	34	112	951	87.02%	20.00%	96.55%	45.16%	89.46%	0.277	0.439	0.819
PC4	86	34	92	1187	90.99%	48.32%	97.22%	71.67%	92.81%	0.577	0.685	0.907
PC5	147	108	356	16390	97.27%	29.23%	99.35%	57.65%	97.87%	0.388	0.539	0.955

Sedangan pada Tabel 5 ditunjukan hasil eksperimen metode Logistic Regression dengan *Resampling* untuk 10 dataset NASA MDP. Hasil eksperimen disajikan adalah akurasi, *sensitivty* (*recall*), *spesificity, Positive Predictive Value* (PPV) atau *Precision*, *Negative predictive Value* (NPV) atau FP_{rate}, *F-Measure*, *G-Mean* dan AUC. Hasil eksperimen pada Tabel 5, menunjukan rata-rata akurasi pada 10 dataset adalah 90.83% dan rata-rata AUC sebesar 0.856.

Tabel 5. Hasil Pengukuran LR dan Resampling

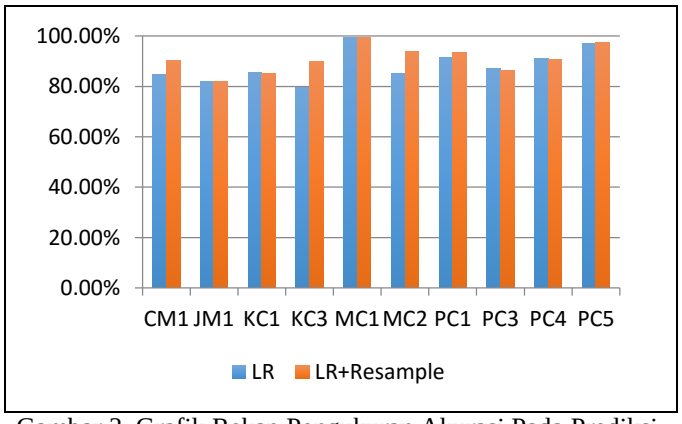
Dataset	TP	TN	FP	FN	Accuracy	Recall	Specificity	PPV	NPV	F-Measure	G-Mean	AUC
CM1	18	10	23	293	90.40%	43.90%	96.70%	64.29%	92.72%	0.522	0.652	0.816
JM1	166	136	1600	7691	81.90%	9.40%	98.26%	54.97%	82.78%	0.161	0.304	0.708
KC1	82	58	253	1703	85.16%	24.48%	96.71%	58.57%	87.07%	0.345	0.487	0.791
KC3	32	15	5	148	90.00%	86.49%	90.80%	68.09%	96.73%	0.762	0.886	0.875
MC1	20	5	43	9209	99.48%	31.75%	99.95%	80.00%	99.54%	0.455	0.563	0.895
MC2	31	12	36	680	93.68%	46.27%	98.27%	72.09%	94.97%	0.564	0.674	0.900
PC1	17	12	38	692	93.41%	30.91%	98.30%	58.62%	94.80%	0.405	0.551	0.854
PC3	29	38	116	942	86.31%	20.00%	96.12%	43.28%	89.04%	0.274	0.438	0.846
PC4	112	43	88	1156	90.64%	56.00%	96.41%	72.26%	92.93%	0.631	0.735	0.926
PC5	143	103	355	16400	97.31%	28.72%	99.38%	58.13%	97.88%	0.384	0.534	0.951

Pada Tabel 6 disajikan rekap pengukuran akurasi model Logistic Regression (LR) dan Logistic Regression dengan penerapan *Resample* (LR+*Resample*).

Tabel 6. Rekap Pengukuran Akurasi LR dan LR+*Resample* pada Prediksi Cacat *Software*

Model	Dataset									
	CM1	JM1	KC1	KC3	MC1	MC2	PC1	PC3	PC4	PC5
LR	84.88%	82.11%	85.74%	79.50%	99.32%	85.04%	91.70%	87.02%	90.99%	97.27%
LR+Resample	90.40%	81.90%	85.16%	90.00%	99.48%	93.68%	93.41%	86.31%	90.64%	97.31%

Dapat dilihat bawah terdapat peningkatan hasil kinerja model dengan melakukan penanganan ketidakseimbangan kelas (*class imbalance*) pada dataset NASA MDP. Perbedaan kinerja yang dihasilkan memang tidak cukup signifikan, yang dapat dilihat pada Gambar 3. Peningkatan kinerja hanya terjadi pada dataset CM1, KC3, dan MC2.



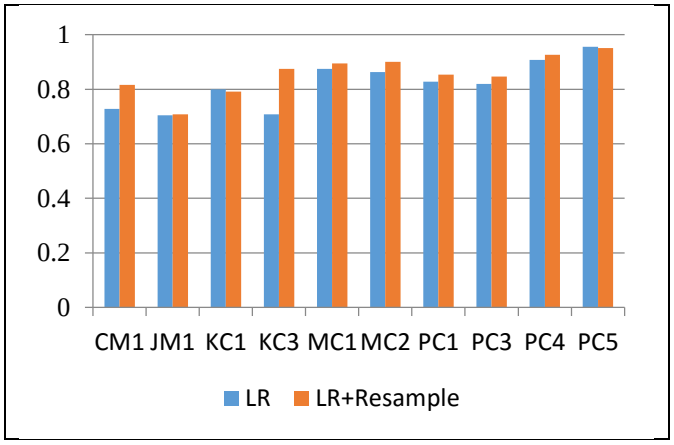
Gambar 3. Grafik Rekap Pengukuran Akurasi Pada Prediksi Cacat Software

Hasil perbandingan *Area Under Curve (*AUC) Logistic Regression (LR) dan Logistic Regression dengan penerapan *Resample* (LR+Resample) disajikan dalam Tabel 7. Dapat dilihat dari Gambar 4 grafik pengukuran AUC mengalami peningkatan kenerja setelah diterapkan metode resample pada dataset yang mengalami ketidakseimbangan kelas dengan peningkatan kinerja pada dataset CM1 dan KC3.

Tabel 7. Rekap Pengukuran AUC Model Prediksi Cacat *Software*

Model	Dataset									
	CM1	JM1	KC1	KC3	MC1	MC2	PC1	PC3	PC4	PC5
LR	0.728	0.705	0.8	0.708	0.875	0.863	0.828	0.819	0.907	0.955
LR+Resample	0.816	0.708	0.791	0.875	0.895	0.9	0.854	0.846	0.926	0.951

Pada penelitian ini dilakukan pengujian hipotesis degan uji *paired sample t-test* untuk Logistic Legression (LR) dengan Logistic Regression dan Resample (LR+*Resample*) dan uji Freidmen (*Friedman test*) untuk membandingkan dengan pengklasifikasi lain. Uji *t (t-test)* adalah hubungan antara variabel respon dengan variable prediktor (Larose, 2005). Hipotesis nol (H_0) menyatakan bahwa tidak ada perbedaan hasil eksperimen antara motode LR dan LR+*Resample*, sedangkan hipotesis satu (H_1) menyatakan bahwa ada perbedaan hasil eksperiment antara LR dan LR+*Resample*.



Gambar 4 Grafik Rekap Pengukuran AUC LR dan LR+Resample pada Prediksi Cacat *Software*

Pada uji *t* sampel berpasangan (*paired-sample t-test)* untuk variabel akurasi LR dan variabel LR+*Resample* dapat dilihat pada Tabel 8.

Tabel 8. *Paired sample t-test* Akurasi LR dan LR+*Resample*

	LR	*LR+Resample*
Mean	88.357	90.829
Variance	40.77077889	29.42509889
Observations	10	10
Pearson Correlation	0.759543832	
Hypothesized Mean Difference	95	
df	9	
t Stat	-73.5139356	
P(T<=t) one-tail	4.03234E-14	
t Critical one-tail	1.833112923	
P(T<=t) two-tail	8.06469E-14	
t Critical two-tail	2.262157158	

Dari hasil uji *t* sampel berpasangan (*paired-sample t-test*) pada Tabel 8 dapat diambil kesimpulan hipotesis berdasarkan perbandingan *t* hitung dan *t* tabel, juga berdasarkan nilai probabilitas. Nilai *t* hitung yang diwakili oleh *t* Stat sebesar 73.513956, dan nilai *t* tabel yang diwakili oleh *t* Critical two-tail sebesar 2.262157158 maka dapat dipastikan nilai *t* hitung > *t* tabel yang artinya H_0 gagal diterima dan H_1 diterima, artinya ada perbedaan antara hasil akurasi LR dan

LR+*Resample*, sedangkan diketahui nilai probabilitas sebesar 8.06469E-14, maka dapat dipatikan bahwa nilai probabilitas < 0,05 yang artinya H_0 gagal diterima dan H_1 diterima, artinya terdapat perbedaan yang signifikan dari rata-rata akurasi LR dan LR+*Resample*, hasil akurasi menunjukan LR+*Resample* lebih tinggi dibandingkan dengan LR.

Selanjutnya uji *t* sampel berpasangan (*paired-sample t-test)* untuk hasil AUC dari LR dan variabel LR+*Resample* dapat dilihat pada Tabel 9. Nilai *t* hitung yang diwakili oleh *t* Stat sebesar 5669.85953, dan nilai *t* tabel yang diwakili oleh *t* Critical two-tail sebesar 2.262157158 maka dapat dipastikan nilai *t* hitung > *t* tabel yang artinya H_0 gagal diterima dan H_1 diterima, artinya ada perbedaan antara hasil AUC LR dan LR+*Resample*, sedangkan diketahui nilai probabilitas sebesar 8.40652E-31, maka dapat dipatikan bahwa nilai probabilitas < 0,05 yang artinya H_0 gagal diterima dan H_1 diterima, artinya terdapat perbedaan yang signifikan dari rata-rata AUC LR dan LR+*Resample*, hasil AUC menunjukan LR+*Resample* lebih tinggi dibandingkan dengan LR.

Tabel 9. Paired sample t-test Akurasi LR dan LR+Resample

	LR	*LR+Resample*
Mean	0.8188	0.8562
Variance	0.007261289	0.005063956
Observations	10	10
Pearson Correlation	0.784614645	
Hypothesized Mean Difference	95	
df	9	
t Stat	-5669.85953	
P(T<=t) one-tail	4.20426E-31	
t Critical one-tail	1.833112923	
P(T<=t) two-tail	8.40852E-31	
t Critical two-tail	2.262157158	

Selanjutnya untuk mengetahui metode yang diusulkan dapat menangani ketidakseimbangan kelas (*class imbalance*) pada dataset NASA maka dibanginkan dengan algoritma pengklasifikasi lain yaitu: Naïve Bayes (NB), Linear Discriminant Analysis (LDA), k-nearest neighbor (k-NN), Decision Tree (C.45), Random Forest (RF), Support Vector Machine (SVM) dan neares neighbor (k*). Eksperimen untuk pengklasifikasi lain menggunakan *software* Rapid Miner versi 5.3. Rekap hasil akurasi dari eksperimen yang dilakukan pada LR, LR+Resample dan pengklasidikasi lain dapat dilihat pada Tabel 10.

Tabel 10. Rekap Akurasi dengan Pengklasifikasi Lain

Model	Dataset									
	CM1	JM1	KC1	KC3	MC1	MC2	PC1	PC3	PC4	PC5
LR	84.88%	82.11%	85.74%	79.50%	99.32%	85.04%	91.70%	87.02%	90.99%	97.27%
LR+R	90.40%	81.90%	85.16%	90.00%	99.48%	93.68%	93.41%	86.31%	90.64%	97.31%
NB	82.56%	81.31%	82.25%	79.00%	93.54%	72.44%	90.93%	33.51%	87.42%	96.63%
LDA	43.02%	81.90%	85.73%	57.00%	97.75%	43.31%	81.03%	65.16%	85.70%	96.58%
KNN	81.69%	70.22%	80.44%	71.50%	99.16%	62.99%	87.35%	80.98%	81.63%	96.82%
C.45	79.07%	81.66%	84.49%	81.50%	99.40%	61.42%	91.83%	87.29%	87.28%	97.11%
SVM	87.50%	80.91%	85.35%	82.00%	99.27%	70.08%	91.83%	87.56%	89.42%	97.30%
RF	88.05%	81.67%	84.49%	82.00%	99.29%	66.14%	91.87%	87.56%	87.28%	97.10%
K*	87.79%	81.66%	84.49%	82.00%	99.27%	65.35%	91.96%	87.56%	12.72%	97.04%

Dari Tabel 10 rekap akurasi dipergunakan untuk uji Friedman (*Friedmant test*) yang menguji hipotesis nol (H_0) menyatakan bahwa tidak ada perbedaan hasil eksperimen antara motode Logitic Regression (LR), Logistic Regression menggunakan *Resample* (LR+R*)*, Naïve Bayes (NB), Linear Discriminant Analysis (LDA), k-nearest neighbor (k-NN), Decision Tree (C.45), Random Forest (RF), Support Vector Machine (SVM) dan neares neighbor (k*). Sedangkan hipotesis satu (H_1) menyatakan bahwa ada perbedaan hasil eksperiment antara LR, LR+R, NB, LDA, KNN, C.45, RF, SVM dan k*. Uji *Friedman* dilakukan menggunakan aplikasi XLSTAT untuk hasil akurasi dari semua pengklasifikasi. Tabel 11 merupakan hasil dari uji *Friedman* untuk *pairwise differences*

Dari hasil uji Friedman model klasifikasi terbaik pada semua dataset dicatak tebal. Dalam uji Friedman didapatkan hasil signifikansi statistik pengujian P-value seperti telihat pada Tabel 11. Berdasarkan *p-value* dapat menjawab hipotesa H_0 diterima jika *p-value* signifikan dan menolak H_1 ketika p-value kurang signifikan. Dalam penelitian ini akan diatur seberapa signifikan statistik dengan symbol α dengan nilai 0.05. Sehingga dapat dirumuskan jika p-value = α menerima H_0 dan menolak H_1.

Tabel 11. Hasil Akurasi Uji Friedman untuk Pairwise Differences

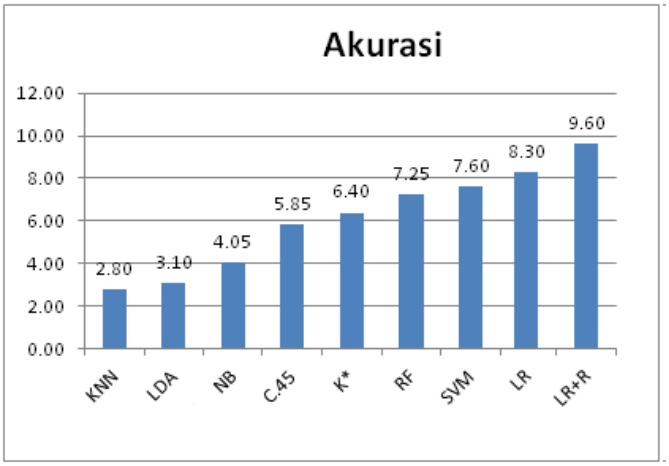
	LR	LR+R	NB	LDA	KNN	C.45	SVM	RF	K*
LR	0	-1.300	4.250	5.200	5.500	2.450	0.700	1.050	1.900
LR+R	1.300	0	5.550	6.500	6.800	3.750	2.000	2.350	3.200
NB	-4.250	-5.550	0	0.950	1.250	-1.800	-3.550	-3.200	-2.350
LDA	-5.200	-6.500	-0.950	0	0.300	-2.750	-4.500	-4.150	-3.300
KNN	-5.500	-6.800	-1.250	-0.300	0	-3.050	-4.800	-4.450	-3.600
C.45	-2.450	-3.750	1.800	2.750	3.050	0	-1.750	-1.400	-0.550
SVM	-0.700	-2.000	3.550	4.500	4.800	1.750	0	0.350	1.200
RF	-1.050	-2.350	3.200	4.150	4.450	1.400	-0.350	0	0.850
K*	-1.900	-3.200	2.350	3.300	3.600	0.550	-1.200	-0.850	0

Untuk uji Friedman ini, kita mengatur tingkat signifikansi statistik (α) menjadi 0,05. Ini berarti bahwa ada perbedaan yang signifikan secara statistik jika P-value <0,05. Dari hasil percobaan, P-value adalah 0,0001, ini lebih rendah dari tingkat signifikansi α = 0,05, dengan demikian kita harus menolak hipotesis nol, dan itu berarti bahwa ada perbedaan yang signifikan secara statistik. Selanjutnya dilakukan uji Friedman dengan *Nemenyi post hoc tes* untuk mendeteksi pengklasifikasi tertentu berbeda secara signifikan. Tabel 12 memperlihatakan hasil akurasi uji Friedman untuk *p-value.*

Tabel 12. Hasil Akurasi Uji Friedman untuk P-values

	LR	LR+R	NB	LDA	KNN	C.45	SVM	RF	K*
LR	1	0.999	0.134	**0.020**	**0.010**	0.860	1.000	1.000	0.972
LR+R	0.999	1	**0.008**	**0.001**	**0.000**	0.288	0.960	0.889	0.536
NB	0.134	**0.008**	1	1.000	0.999	0.981	0.371	0.536	0.889
LDA	**0.020**	**0.001**	1.000	1	1.000	0.747	0.086	0.159	0.487
KNN	**0.010**	**0.000**	0.999	1.000	1	0.609	**0.047**	0.094	0.349
C.45	0.860	0.288	0.981	0.747	0.609	1	0.985	0.997	1.000
SVM	1.000	0.960	0.371	0.086	**0.047**	0.985	1	1.000	0.999
RF	1.000	0.889	0.536	0.159	0.094	0.997	1.000	1	1.000
K*	0.972	0.536	0.889	0.487	0.349	1.000	0.999	1.000	1

Dapat dilihat pada Gambar 5, Logistic Regression dengan *Resampling* (LR+R*)* menghasilkan akurasi tinggi berdasarkan uji *Freidmen.* Berdasarkan Tabel 12. dapat ditunjukan bahwa pengklasifiasi Naïve Bayes (NN) danLogistic Regression (LR) menghasilkan akurasi yang tinggi seperti penelitian oleh (Hall et al., 2012; Wahono et al., 2011). Seperti yang dilakukan oleh (Saifudin, 2014) yang menggunakan dataset NASA, hasil akurasi NB dan LR menunjukan hasil yang signifikan.



Gambar 5. Grafik Mean Akurasi Uji Friedman

Pada uji Friedman selanjutnya adalah menguji hasil AUC dari eksperimen yang sudah dilakukan. Tabel 13 menujukan rekap hasil eksperimen AUC. Dari Tabel 13 rekap AUC dipergunakan untuk uji Friedman (*Friedman test*) yang menguji hipotesis nol (H_0) menyatakan bahwa tidak ada perbedaan hasil eksperimen antara motode LR, LR+R, NB, LDA, k-NN, C.45, RF, SVM dan k*. Sedangkan hipotesis satu (H_1) menyatakan bahwa ada perbedaan hasil eksperiment antara LR, LR+R, NB, LDA, KNN, C.45, RF, SVM dan k*. Uji *Friedman* dilakukan menggunakan aplikasi XLSTAT untuk hasil akurasi dari semua pengklasifikasi. Tabel 11 merupakan hasil dari uji *Friedman* untuk *pairwise differences*

Tabel 13. Perbandingan AUC Model Prediksi Cacat Software

Model	Dataset									
	CM1	JM1	KC1	KC3	MC1	MC2	PC1	PC3	PC4	PC5
LR	0.728	0.705	0.800	0.708	0.875	0.863	0.828	0.819	0.907	**0.955**
LR+R	0.816	0.708	0.791	0.875	0.895	**0.900**	0.854	0.846	**0.926**	0.951
NB	0.780	0.683	0.786	0.677	**0.916**	0.712	0.775	0.756	0.840	0.940
LDA	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500
KNN	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500
C.45	0.500	0.500	0.542	0.541	0.836	0.489	0.609	0.696	0.723	0.500
SVM	0.736	0.614	0.731	0.595	0.509	0.716	0.810	0.732	0.905	0.784
RF	0.518	0.000	0.549	0.599	0.639	0.627	0.523	0.509	0.150	0.676
K*	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500

Dari hasil AUC uji Friedman model klasifikasi terbaik pada semua dataset dicetak tebal. Dalam uji Friedman didapatkan hasil signifikansi statistik pengujian P-value seperti telihat pada Tabel 14. Berdasarkan *p-value* dapat menjawab hipotesa H_0 diterima jika *p-value* signifikan dan menolak H_1 ketika p-value kurang signifikan. Dalam penelitian ini akan diatur seberapa signifikan statistik dengan symbol α dengan nilai 0.05. Sehingga dapat dirumuskan jika p-value = α menerima H_0 dan menolak H_1.

Tabel 14. Hasil AUC Uji Friedman untuk *Pairwise Differences*

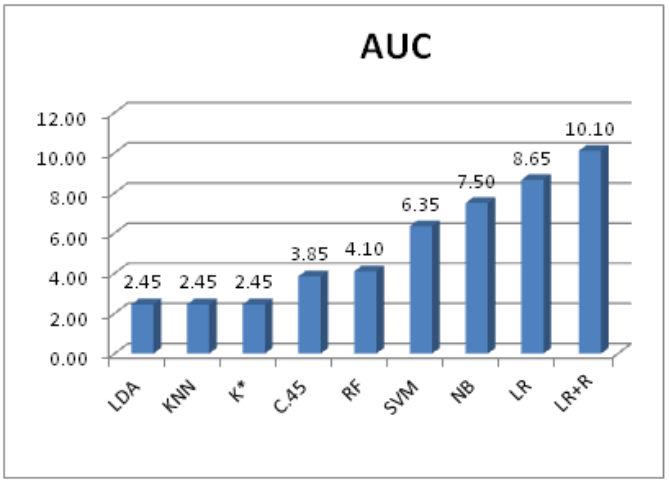
	LR	LR+R	NB	LDA	KNN	C.45	SVM	RF	K*
LR	0	-1.450	1.150	6.200	6.200	4.800	2.300	4.550	6.200
LR+R	1.450	0	2.600	7.650	7.650	6.250	3.750	6.000	7.650
NB	-1.150	-2.600	0	5.050	5.050	3.650	1.150	3.400	5.050
LDA	-6.200	-7.650	-5.050	0	0.000	-1.400	-3.900	-1.650	0.000
KNN	-6.200	-7.650	-5.050	0.000	0	-1.400	-3.900	-1.650	0.000
C.45	-4.800	-6.250	-3.650	1.400	1.400	0	-2.500	-0.250	1.400
SVM	-2.300	-3.750	-1.150	3.900	3.900	2.500	0	2.250	3.900
RF	-4.550	-6.000	-3.400	1.650	1.650	0.250	-2.250	0	1.650
K*	-6.200	-7.650	-5.050	0.000	0.000	-1.400	-3.900	-1.650	0

Untuk uji Friedmant ini, kita mengatur tingkat signifikansi statistik (α) menjadi 0,05. Ini berarti bahwa ada perbedaan yang signifikan secara statistik jika P-value <0,05. Dari hasil percobaan, P-value adalah 0,0001, ini lebih rendah dari tingkat signifikansi α = 0,05, dengan demikian kita harus menolak hipotesis nol, dan itu berarti bahwa ada perbedaan yang signifikan secara statistik. Selanjutnya dilakukan uji Friedman dengan *Nemenyi post hoc tes* untuk mendeteksi pengklasifikasi tertentu berbeda secara signifikan. Tabel 15 memperlihatakan hasil akurasi uji Friedman untuk *p-value.*

Tabel 15. Hasil AUC Uji Friedman untuk P-values

	LR	LR+R	NB	LDA	KNN	C.45	SVM	RF	K*
LR	1	0.997	1.000	**0.001**	**0.001**	**0.047**	0.902	0.078	**0.001**
LR+R	0.997	1	0.808	**0.0001**	**0.0001**	**0.001**	0.288	**0.003**	**0.0001**
NB	1.000	0.808	1	**0.028**	**0.028**	0.328	1.000	0.439	**0.028**
LDA	**0.001**	**0.0001**	**0.028**	1	1.000	0.997	0.233	0.990	1.000
KNN	**0.001**	**0.0001**	**0.028**	1.000	1	0.997	0.233	0.990	1.000
C.45	**0.047**	**0.001**	0.328	0.997	0.997	1	0.843	1.000	0.997
SVM	0.902	0.288	1.000	0.233	0.233	0.843	1	0.915	0.233
RF	0.078	**0.003**	0.439	0.990	0.990	1.000	0.915	1	0.990
K*	**0.001**	**0.0001**	**0.028**	1.000	1.000	0.997	0.233	0.990	1

Dapat dilihat pada Gambar 6, Logistic Regression dengan *Resampling* (LR+R) menghasilkan AUC tinggi berdasarkan uji *Freidmen.* Berdasarkan Tabel 15, dapat ditunjukan bahwa pengklasifiasi Naïve Bayes (NN) dan Logistic Regression (LR) menghasilkan AUC yang tinggi seperti penelitian oleh (Hall et al., 2012; Wahono et al., 2011). Seperti yang dilakukan oleh (Saifudin, 2014) yang menggunakan dataset NASA, hasil akurasi NB dan LR menunjukan hasil yang signifikan.



Gambar 6. Grafik Mean AUC uji Friedman

5 KESIMPULAN

Penelitian dengan menerapkan metode *resampling* untuk penyelesaian ketidakseimbangan kelas (*class Imbalance*) pada dataset NASA MDP untuk prediksi cacat *Software* dengan algoritma Logistic Regression. Hasil eksperimen pada penelitian ini mendapatakn nilai akurasi sebesar 99,48% pada dataset MC1 dengan model LR+*Resample*, mengalami peningkatan sebesar 0.16% dari LR tanpa *Resample.* Dan hasil AUC sebesar 0.951 pada dataset PC5 untuk model LR+*Resample.*

Hasil perbandingan dari eksperimen pada penelitian untuk semua dataset dengan pengklasifikasi lain (Naive Bayes (NB), Linear Discriminant Analysis (LDA), k-Nearest Neghbor (k-NN), C4.5, Support Vector Machine (SVM), Random Forest (RF), dan K*) tingkat akurasi Logistic Regression menunjukan hasil yang paling baik, baik dalam parameter AUC maupun akurasi.

Dari hasil pengujian di atas maka dapat disimpulkan bahwa pengunaan metode LR+*Resample* mampu menangani ketidakseimbangan dataset pada logistic regression dengan menghasilkan nilai akurasi dan AUC lebih tinggi dibandingkan dengan metode LR yang tidak menggunakan Resample. Dari hasil pengujian diatas dapat disimpulkan bahwa metode *resampling* terbukti efektif dalam menyelesaikan ketidakseimbangan kelas pada prefiksi cacat *Software* dengan algoritma Logistic Regression.

REFERENSI

Alpaydin, E. (2010). *Introduction to Machine Learning*. London: The MIT Press.

Canu, S., & Smola, A. (2006). Kernel methods and the exponential family. *Neurocomputing*.

Chang, R., Mu, X., & Zhang, L. (2011). Software Defect Prediction Using Non-Negative Matrix Factorization. *Journal of Software*.

Czibula, G., Marian, Z., & Czibula, I. G. (2014). Software defect prediction using relational association rule mining. *Information Sciences*.

Dubey, R., Zhou, J., Wang, Y., Thompson, P. M., & Ye, J. (2014). Analysis of sampling techniques for imbalanced data: An n=648 ADNI study. *NeuroImage*.

Ganganwar, V. (2012). An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering*.

Gorunescu, F. (2011). Data mining: Concepts, models and techniques. *Intelligent Systems Reference Library*, *12*.

Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A Systematic Literature Review on Fault Prediction Performance in Software Engineering. *IEEE Transactions on Software Engineering*.

Harrington, P. (2012). *Machine Learning in Action*. Manning Publications Co.

Hosmer, D. W., & Lemeshow, S. (2000). *Applied Logistic Regression Second Edition*. New York, NY: John Wiley & Sons, Inc.

Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied Logistic Regression Third Edition*. Hoboken, NJ, USA: John Wiley & Sons, Inc.

Karsmakers, P., Pelckmans, K., & Suykens, J. a. K. (2007). Multi-class kernel logistic regression: a fixed-size implementation. *2007 International Joint Conference on Neural Networks*.

Khoshgoftaar, T. M., Gao, K., Napolitano, A., & Wald, R. (2013). A comparative study of iterative and non-iterative feature selection techniques for software defect prediction. *Information Systems Frontiers*.

Komarek, P., & Moore, A. W. (2005). Making Logistic Regression A Core Data Mining Tool. *School of Computer Science*.

Larose, D. T. (2005). *Discovering Knowladge In Data: An Introduction to Data Mining*. *Discovering Knowledge in Data: An Introduction to Data Mining*.

Lessmann, S., Member, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking Classification Models for Software Defect Prediction : A Proposed Framework and Novel Findings.

Liebchen, G. a., & Shepperd, M. (2008). Data sets and data quality in software engineering. *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*.

Lin, C., Weng, R. C., & Keerthi, S. S. (2008). Trust Region Newton Method for Large-Scale Logistic Regression. *Journal of Machine Learning Research*.

Ma, Y., Luo, G., Zeng, X., & Chen, A. (2012). Transfer learning for cross-company software defect prediction. *Information and Software Technology*.

Maalouf, M., & Trafalis, T. B. (2011). Robust weighted kernel logistic regression in imbalanced and rare events data. *Computational Statistics & Data Analysis*.

Song, Q., Jia, Z., Shepperd, M., Ying, S., & Liu, J. (2011). A General Software Defect-Proneness Prediction Framework. *IEEE Transactions on Software Engineering*.

Thanathamathee, P., & Lursinsap, C. (2013). Handling imbalanced data sets with synthetic boundary data generation using bootstrap re-sampling and AdaBoost techniques. *Pattern Recognition Letters*.

Vercellis, C. (2011). *Business Intelligence: Data Mining and Optimization for Decision Making*. *Methods*. John Wiley & Sons.

Wahono, R. S., Suryana, N., & Ahmad, S. (2014). Metaheuristic Optimization based Feature Selection for Software Defect Prediction. *Journal of Software*.

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data Mining Third Edition*. Elsevier Inc.

Wu, J., & Cai, Z. (2011). Attribute Weighting via Differential Evolution Algorithm for Attribute Weighted Naive Bayes (WNB).

Wu, X., & Kumar, V. (2010). *The Top Ten Algorithms in Data Mining*. Taylor & Francis Group.

BIOGRAFI PENULIS

Harsih Rianto. Memperoleh gelar S.Kom dari Sekolah Tinggi Ilmu Komputer Nusa Mandiri Jakarta (STMIK Nusa Mandiri) dan M.Kom dari program pasca sarjana program studi Magister Ilmu Komputer STMIK Nusa Mandiri, Jakarta. Saat ini bekerja sebagai dosen di AMIK BSI Bekasi. Minat penelitiannya saat ini meliputi mechine learning dan software engineering (rekayasa perangkat lunak).

Romi Satria Wahono. Memperoleh gelar B.Eng dan M.Eng pada bidang ilmu komputer di Saitama University Japan, dan Ph.D pada bidang software engineering di Universiti Teknikal Malaysia Melaka. Pengajar dan peneliti di Fakultas Ilmu Komputer, Universitas Dian Nuswantoro. Pendiri dan CEO PT Brainmatics, perusahaan yang bergerak di bidang pengembangan software. Minat penelitian pada bidang software engineering dan machine learning. Profesional member dari asosiai ilmiah ACM, PMI dan IEEE Computer Society.

Estimasi Proyek Pengembangan Perangkat Lunak dengan Fuzzy Use Case Points

Muhadi Hariyanto
Program Studi Sistem Informasi, STMIK Nusa Mandiri
muhadi.mho@nusamandiri.ac.id

Romi Satria Wahono
Fakultas Ilmu Komputer, Universitas Dian Nuswantoro
romi@romisatriawahono.net

Abstract: Perangkat lunak memegang peranan penting agar sebuah komputer atau sistem dapat digunakan, sehingga dibutuhkan manajemen proyek dalam pengembangan perangkat lunak dan salah satu prosesnya adalah melakukan estimasi agar perangkat lunak yang dihasilkan sesuai dengan jadwal dan biaya yang telah ditentukan. Metode use case points banyak dipakai terutama untuk aplikasi yang berbasis obyek, tetapi ditemukan beberapa kelemahan berupa ketidakpastian faktor biaya dan penentuan klasifikasi use case memiliki beda nilai cukup tinggi yang mengakibatkan hasil estimasi kurang akurat. Metode use case points dimodifikasi dengan menambahkan logika fuzzy sehingga menjadi metode fuzzy use case points, komponen yang dimodifikasi yaitu pada penilaian klasifikasi use case. Modifikasi yang dilakukan berupa penentuan nilai use case berdasarkan jumlah transaksi. Dari hasil percobaan yang dilakukan, nilai effort dari metode ini lebih akurat atau mendekati effort aktual. Peningkatan akurasi metode ini mencapai 6 sampai 10%.

Keywords: estimasi proyek software, fuzzy use case point

1. PENDAHULUAN

Perangkat lunak merupakan abstraksi fisik yang memungkinkan kita untuk berbicara dengan mesin perangkat keras (Langer, 2008). Tanpa adanya perangkat lunak, maka perangkat keras yang telah diciptakan tidak akan dapat berguna atau berfungsi dengan optimal.

Untuk menghasilkan perangkat lunak yang berkualitas, dalam pengembangannya perlu adanya manajemen proyek dengan mengikuti pola SDLC (*Software Development Life Cycle*). Salah satu kegiatannya adalah melakukan estimasi. Metode estimasi yang dapat digunakan *Simply Method (Industri Information Based)* (Dennis, 2005), *Function Points Approach* (Albrecht, 1979), dan *Use Case Points* (Karner, 1993).

Penggunaan metode *use case points* memberikan perkiraan *effort* mendekati perkiraan sebenarnya yang dihasilkan oleh pengembang perangkat lunak berpengalaman (Anda, 2002). Ini menandakan pengalaman tim proyek akan mempengaruhi tingkat akurasi estimasi yang dilakukan. *Use case points* juga memiliki kelemahan berupa ketidakpastian faktor biaya dan penentuan klasifikasi yang memiliki beda nilai pengali cukup tinggi (Fan, 2009). Keterbatasan atau kelemahan yang akan mempengaruhi keakuratan estimasi yaitu adanya perbedaan nilai pengali cukup tinggi dari level kompleksitas tiap *use case* (Nassif, 2010).

Estimasi proyek perangkat lunak dengan menggunakan *use case points* mudah digunakan tetapi menjadi kurang akurat karena penentuan kompleksitas *use case* dianggap bersifat linier bila dilihat dari jumlah transaksi setiap *use case*. Sebagai alternatif, digunakan *fuzzy use case points* yang merupakan modifikasi dari *use case points* yaitu dengan menambahkan atau memodifikasi nilai pengali dari klasifikasi jumlah transaksi yang ada di *use case* dengan menggunakan logika *fuzzy*.

2. PENELITIAN TERKAIT

Mohammed Wajahat Kamal dan Moataz A. Ahmed melakukan perbandingan metode penghitungan *effort* dalam proyek pengembangan perangkat lunak di tahun 2011. Latar belakang masalah yang mendasari mereka melakukan penelitian tersebut karena terdapat beberapa metode yang dapat dilakukan untuk menghitung estimasi *effort* di awal proyek, tetapi akan sulit dilakukan karena data awal kurang akurat dan kurang detail. Mereka melakukan perbandingan beberapa metode, yaitu: *Use Case Points, Transactions, Paths, Extended Use Case Points, UCPm, Adapted Use Case Points, Use Case Size Points, Fuzzy Use Case Points, Simplified Use Case Points,* dan *Industrial use of Use Case Points*. Dari hasil pengujian metode yang ada, ternyata *Fuzzy Use Case Points* mampu menangani informasi yang tidak atau kurang tepat dan transparansi.

Mohammad Saber Iraji, Majid Aboutalebi dan Homayun Motameni di tahun 2011 melakukan penelitian untuk menemukan metode yang secara otomatis dapat menentukan tingkat kompleksitas *use case* pada saat melakukan estimasi *effort* dengan menggunakan *use case points*. Latar belakang masalah yang mendasari karena metode *use case points* memiliki kelemahan berupa ketidakpastian dari faktor biaya dan penentuan klasifikasinya kurang detail. Metode yang disarankan oleh mereka adalah menggunakan *Neuro Fuzzy Use Case Points* (NFUCP), dan dari hasil pengujian menunjukkan kalau memang pengolahan data dengan menggunakan NFUCP lebih akurat dan mendekati *actual effort* dari pengembangan sebuah perangkat lunak.

Ali Bou Nasif, Luiz Fernando Capretz dan Danny Ho melakukan penelitian untuk meningkatkan keakuratan estimasi metode *Use Case Points* dengan teknik *Soft Computing* di tahun 2010. Latar belakang penelitian karena identifikasi dari *International Society of Parametrics Analysis* dan *The Standish Group International* bahwa estimasi yang jelek sebagai salah satu penyebab dari kegagalan pengembangan perangkat lunak. Mereka melakukan modifikasi *Use Case Points* dengan *Fuzy Logic*

untuk menghitung faktor kompleksitas *Use Case* dan menggunakan *Neural Network* untuk memetakan data input berupa *use case* dan aktor yang terlibat. Dari pengujian yang mereka lakukan, optimasi *Use Case Points* dengan *Fuzzy Logic* dan *Neural Network* meningkatkan keakuratan sampai 22% pada beberapa proyek.

3. LANDASAN TEORI

3.1 Use Case Points

Use case adalah cara formal yang menggambarkan bagaimana sebuah sistem bisnis berinteraksi dengan lingkungannya (Dennis, 2005). *Use Case Points* dikembangkan oleh Gustav Karner di tahun 1993 (Karner, 1993). Untuk melakukan estimasi proyek menggunakan *use case points*, dibuat dahulu sebuah *use case diagram*. Kemudian klasifikasikan aktor sesuai Tabel 1. Hasil perkalian jumlah aktor sesuai klasifikasi aktor disebut *Unadjusted Actor Weight* (UAW).

Tabel 1. *Unadjusted Actor Weighting Table* [2]

Actor Type	*Description*	*Weighting Factor*
Simple	*External System with well-defined API*	1
Average	*External system using a protocol based interface*	2
Complex	*Human*	3

Selain aktor, setiap *use case* diklasifikasikan berdasarkan jumlah transaksinya. Hasil perhitungan jumlah *use case* dan klasifikasinya disebut *Unadjusted Use Case Weight (UUCW)*.

Tabel 2. *Unadjusted Use Case Weighting Table* [2]

Use Case	*Description (Transactions)*	*Weighting Factor*
Simple	1 – 3	5
Average	4 – 7	10
Complex	> 7	15

Nilai UAW dan UUCW dijumlahkan menjadi nilai *Unadjusted Use Case Points (UUCP)*. Selain klasifikasi aktor dan *use case*, ada dua jenis faktor yang akan dihitung yaitu *Technical Complexity Factor* dan *Environmental Factor*. Setiap faktor memiliki beban atau nilai pengali yang berbeda-beda. Nilai awal yang diberikan untuk tiap faktor berkisar dari 0 – 5 dengan kriteria diberi nilai 0 apabila faktor tersebut tidak relevan dan 5 jika sangat penting. Sedangkan bila faktor tidak penting dan tidak relevan maka diberi nilai 3.

Tabel 3. *Technical Complexity Factors Table* [2]

No	*Description*	*Weight*
T_1	*Distributed Systems*	2.0
T_2	*Response time or throughput performance objectives*	1.0
T_3	*End-user online efficiency*	1.0
T_4	*Complex internal proccesing*	1.0
T_5	*Reusability of code*	1.0
T_6	*Easy to install*	0.5
T_7	*Ease of use*	0.5
T_8	*Portability*	2.0
T_9	*Ease of change*	1.0
T_{10}	*Concurrency*	1.0
T_{11}	*Special security objectives included*	1.0
T_{12}	*Direct access for third parties*	1.0
T_{13}	*Special user training required*	1.0

Tabel 3 digunakan untuk menghitung nilai *Technical Complexity Factor (TCF)* dengan rumus:

$$TCF = 0.6 + \left(0.01 \sum_{i=1}^{13} F_i * W_i\right)$$

Tabel 4. *Environmental Factors Table* [2]

No	*Description*	*Weight*
E_1	*Familiarity with system development process being used*	1.5
E_2	*Application experience*	0.5
E_3	*Object-oriented experience*	1.0
E_4	*Lead analyst capability*	0.5
E_5	*Motivation*	1.0
E_6	*Requirements stability*	2.0
E_7	*Part time staff*	-1.0
E_8	*Difficulty of programming language*	-1.0

Tabel 4 digunakan untuk menghitung nilai *Environmental Factor (EF)* dengan rumus:

$$EF = 1.4 + \left(-0.03 \sum_{i=1}^{8} F_i * W_i\right)$$

Nilai UUCP, TCF dan EF digunakan untuk menghitung UCP atau *Unadjusted Case Points* dengan rumus:

$$UCP = UUCP * TCF * EF$$

Hasil UCP sudah menggambarkan ukuran dari sistem yang akan dibuat, langkah selanjutnya menghitung berapa banyak *effort* atau tenaga yang dibutuhkan. Nilai *effort* dapat diketahui dari hasil UCP dibagi dengan nilai PHM (*Person Hour Multiplier*). Nilai PHM diperoleh dengan aturan sebagai berikut:

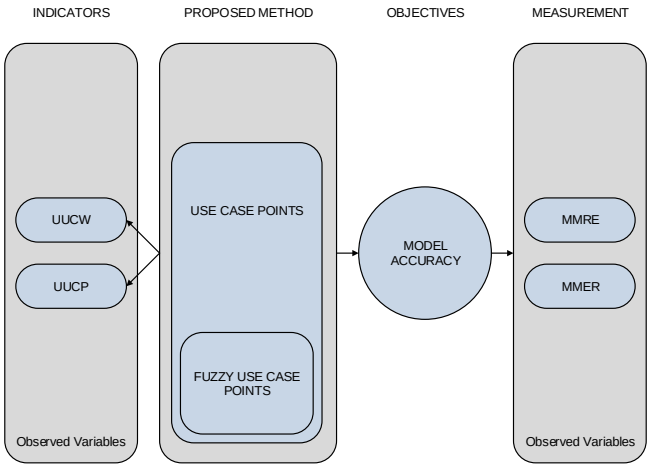
- F_1 = Jumlah E_1 sampai E_6 yang < 3
- F_2 = Jumlah E_7 sampai E_8 yang > 3
- Jika $F_1 + F_2$ <= 2 maka PHM = 20
- Jika $F_1 + F_2$ = 3 atau 4 maka PHM = 28
- Jika $F_1 + F_2$ > 4 maka proyek harus dibatalkan

3.2 Fuzzy Logic

Konsep logika *fuzzy* diperkenalkan oleh Prof. Lotfi Zadeh dari Universitas California di Berkeley pada 1965, dipresentasikan bukan sebagai metodologi kontrol, tetapi sebagai suatu cara pemrosesan data dengan memperkenankan penggunaan *partial set membership* dibanding *crisp set membership*.

4 METODE PENELITIAN

Kerangka pemikiran penelitian ini, seperti ditampilkan dalam Gambar 1, dimulai dari prediksi *effort* dalam proyek perangkat lunak. Penelitian ini menggunakan data enam *project*. Metode yang diusulkan adalah menggunakan *use case points* yang dimodifikasi dengan bantuan logika *fuzzy*.

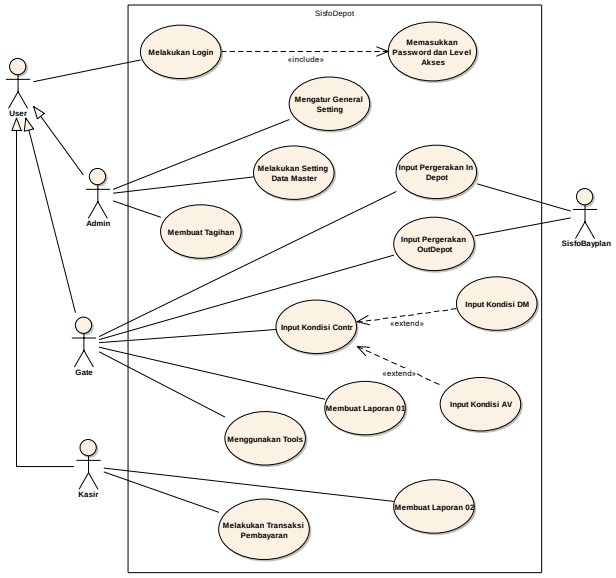


Gambar 1. Kerangka Pemikiran Penelitian

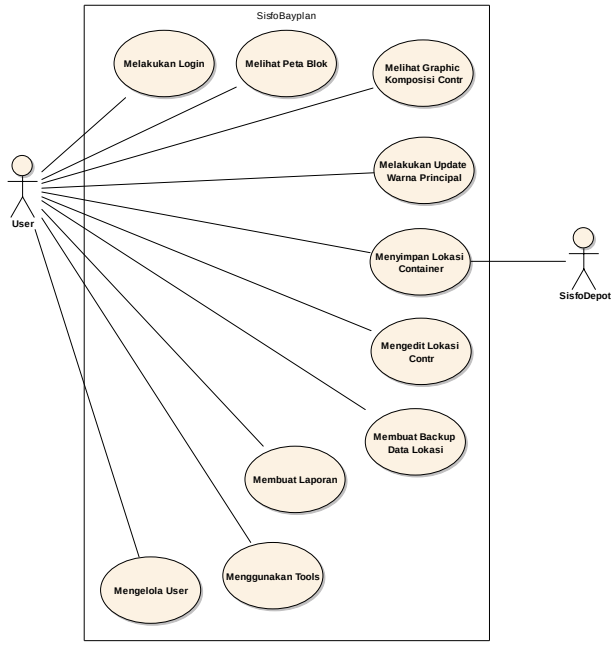
Data dalam penelitian ini diperoleh dengan menggunakan teknik wawancara ke *programmer* dari proyek perangkat lunak yang sedang atau sudah selesai dikerjakan. Data yang didapat berupa *use case metrics* dari enam proyek perangkat lunak.

Semua proyek memiliki *effort* aktual dalam satuan *hours*, nilai *effort* tersebut merupakan jumlah waktu yang diperlukan oleh *programmer* dalam menyelesaikan proyeknya, mulai dari tahap *planning*, *analysis*, *design*¸ dan *implementation*. Berikut ini diagram *use case* dari enam proyek yang dijadikan bahan penelitian.

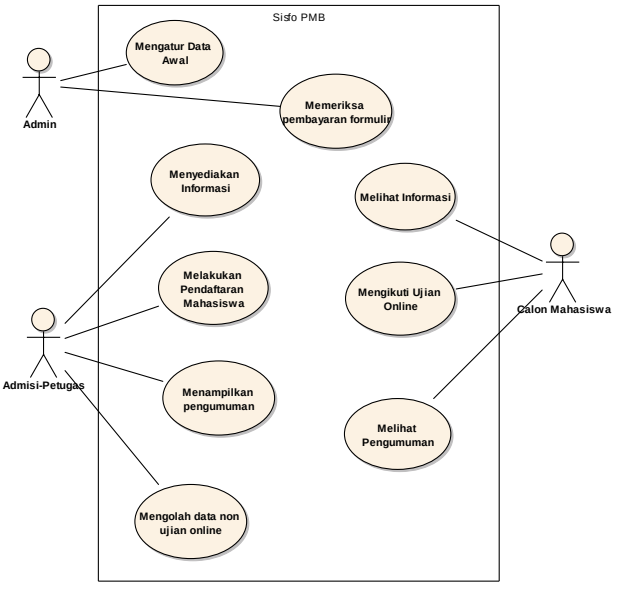
Gambar 2. Diagram *Use Case* Sistem Informasi Sekolah



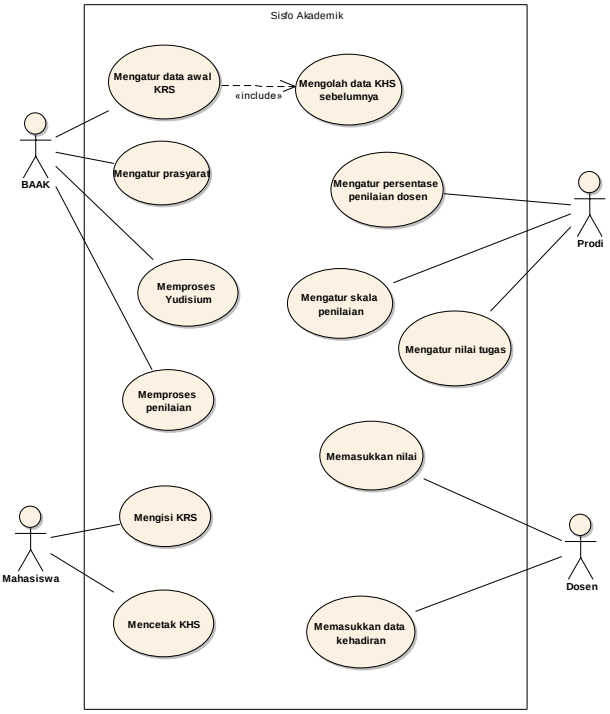
Gambar 3. Diagram *Use Case* Sistem Informasi Depot



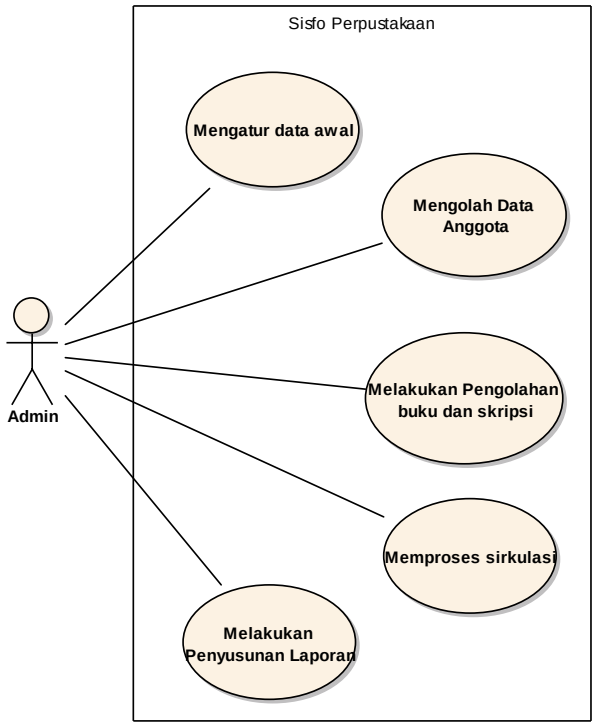
Gambar 4. Diagram *Use Case* Sistem Informasi Bayplan



Gambar 5. Diagram *Use Case* Sistem Informasi Penerimaan Mahasiswa Baru



Gambar 6. Diagram *Use Case* Sistem Informasi Akademik



Gambar 7. Diagram *Use Case* Sistem Informasi Perpustakaan

5 METODE YANG DIUSULKAN

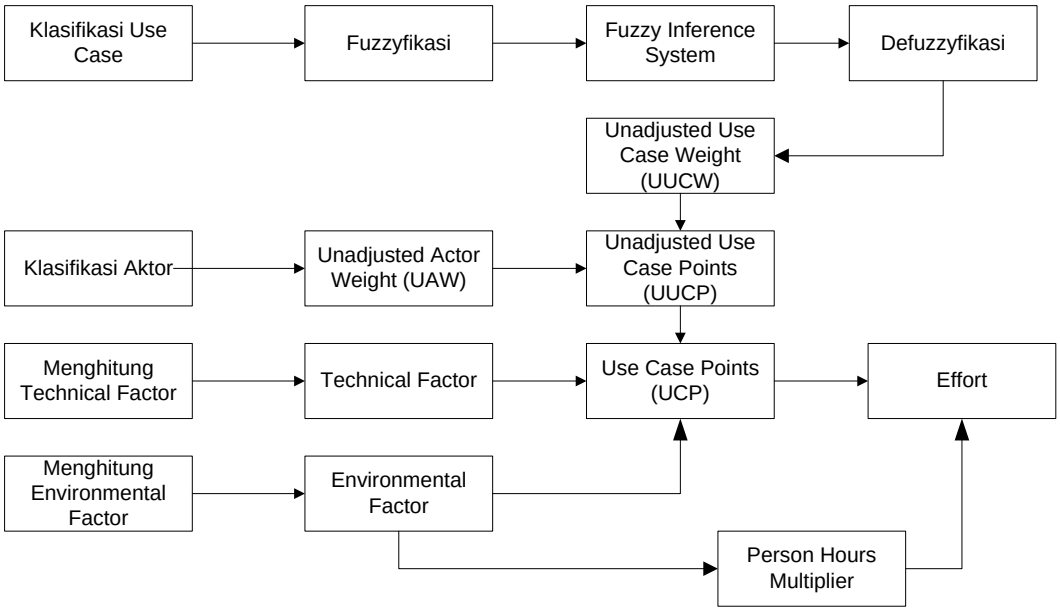
Metode yang diusulkan adalah dengan menyempurnakan menghitung kompleksitas *use case* dengan menggunakan *fuzzy inference system* metode Mamdani karena memiliki keuntungan lebih intuitif, memiliki penerimaan luas dan cocok untuk *input* manusia (Sivanandam, 2007). Sehingga dengan demikian, *fuzzy inference system* yang dibuat menjadi lebih mudah dalam penggunaan dan pembacaan hasilnya.

Logika *fuzzy* yang dibuat menggunakan representasi kurva segitiga karena bentuk kurva segitiga lebih sederhana dalam penentuan dan mudah divisualisasikan, selain itu dalam pemodelan sistem dinamis dengan menggunakan fungsi segitiga dapat memperkirakan perilaku atau nilai yang hampir presisi di setiap tingkat (Cox, 1994). Variabel masukan terdiri dari tiga kurva segitiga untuk memasukkan jumlah transaksi dari setiap *use case*. Variabel keluaran terdiri dari tiga kurva segitiga yang menghasilkan nilai pengali dari tiap *use case*.

Gambar 8 menunjukkan ada empat kategori data yang digunakan sebagai masukan dalam menghitung estimasi dan akan menghasilkan satu keluaran berupa *effort*. Hanya satu kategori yaitu klasifikasi aktor yang menggunakan *fuzzy inference system*, sedangkan kategori yang lain diolah sesuai dengan metode *use case points*.

Penelitian menggunakan model eksperimen yang menyelidiki hubungan kausal menggunakan data yang dikendalikan sendiri oleh peneliti. *Fuzzy inference system* yang dibuat hanya terdiri dari satu variabel masukan dan satu variabel keluaran dengan menggunakan representasi kurva segitiga.

Klasifikasi *use case* dengan menggunakan metode *use case points* dikelompokkan menjadi tiga yaitu *simple*, *average* dan *complex*. Pada metode yang diusulkan dan hasil pengujian metode diperoleh klasifikasi kompleksitas *use case* menjadi 12 kelompok dengan asumsi jumlah maksimal transaksi pada setiap *use case* adalah 12 transaksi.



Gambar 8. Metode yang Diusulkan

Tabel 5. Perbandingan Nilai Klasifikasi *Use Case*

Transaksi di *use case*	Nilai metode *use case*	Nilai *fuzzy use case*
1	5	5,00
2	5	5,00
3	5	6,45
4	10	7,50
5	10	8,55
6	10	10,00
7	10	11,40
8	15	12,50
9	15	13,60
10	15	15,00
11	15	15,00
12	15	15,00

Metode yang diusulkan pada penelitian ini adalah dengan menerapkan logika *fuzzy* yang digunakan untuk menentukan nilai pengali dari pengklasifikasian *use case* sehingga didapat nilai yang lebih bervariatif dan menghasilkan nilai effort yang lebih mendekati nilai effort sebenarnya.

Penelitian ini menggunakan dua metode untuk membandingkan nilai *effort*, yaitu *Mean Magnitude of Relative Error* (MMRE) yang merupakan kriteria yang sangat umum digunakan untuk mengevaluasi model estimasi *effort* perangkat lunak (Briand, 1998). Nilai MMRE merupakan nilai rata-rata dari sejumlah nilai MRE yang didapat dengan rumus:

$$\boldsymbol{MRE_i = \frac{|Actual\ Effort_i - Predicted\ Effort_i|}{Actual\ Effort_i}}$$

Metode kedua adalah *Mean Magnitude of Error Relative* (MMER) sebagai alternatif metode lain untuk mengevaluasi model estimasi *effort* perangkat lunak (Kitchenham, 2001). Nilai MMER merupakan nilai rata-rata dari sejumlah nilai MER yang didapat dengan rumus:

$$\boldsymbol{MER_i = \frac{|Actual\ Effort_i - Predicted\ Effort_i|}{Predicted\ Effort_i}}$$

Ketika menggunakan MMRE dan MMER untuk evaluasi, hasil yang bagus ditunjukkan dengan nilai yang rendah. Terkadang hasil dari MMRE dan MMER kurang akurat sehingga perlu ditambah dengan *Mean Error with Standard Deviation* (Nassif, 2010). Langkah pertama mencari nilai rerata error setiap metode dengan rumus:

$$\boldsymbol{\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i}$$

Dimana

$$\boldsymbol{x_1 = (Actual\ Effort_i - Predicted\ Effort_i)}$$

Nilai deviasi diperoleh dengan rumus:

$$\boldsymbol{SD = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}}$$

Sedangkan nilai *Mean Error with Standard Deviation* didapat dengan rumus: $\boldsymbol{\bar{x} \pm SD}$

6 HASIL PENELITIAN DAN PEMBAHASAN

6.1 Eksperimen dan Pengujian Metode

6.1.1 Metode *Use Case*

Use case diagram yang terdiri dari *use case* dan aktor mengggambarkan apa yang dapat dikerjakan oleh sistem. Sebagai contoh pengujian, diambil data salah satu proyek perangkat lunak yaitu aplikasi Sistem Informasi Depot.

Dari Gambar 3, diketahui terdapat lima aktor dan 14 *use case*. Tahapan yang dilakukan:

1. Klasifikasikan aktor yang terlibat untuk menghasilkan nilai *Unadjusted Actor Weight* (UAW).

Tabel 6. *Unadjusted Actor Weighting Table*

Actor Type	*Description*	*Weighting Factor*	*Number*	*Result*
Simple	*External System with well-defined API*	1	--	--
Average	*External system using a protocol based interface*	2	1	2
Complex	*Human*	3	4	12
Unadjusted Actor Weight Total (UAW)				14

2. Klasifikasikan *use case* yang ada di *use case* diagram untuk menghasilkan nilai *Unadjusted Use Case Weighting* (UUCW)

Tabel 7. *Unadjusted Use Case Weighting Table*

Use Case Type	*Description*	*Weighting Factor*	*Number*	*Result*
Simple	1 – 3 *transactions*	5	6	30
Average	4 – 7 *transactions*	10	3	30
Complex	> 7 *transactions*	15	5	75
Unadjusted Use Case Weight Total (UUCW)				135

3. Nilai UAW dan UUCW dijumlahkan menjadi nilai *Unadjusted Use Case Points* (UUCP), sehingga UUCP dari aplikasi tersebut adalah 149.
4. Menghitung nilai faktor teknik untuk memperoleh nilai *Technical Complexity Factor* (TCF).

Tabel 8. *Technical Complexity Factors Table*

Factor Number	*Description*	*Weight*	*Assigned Value (0–5)*	*Weighted Value*
T_1	*Distributed Systems*	2,0	5	10,0
T_2	*Response time or throughput performance objectives*	1,0	4	4,0
T_3	*End-user online efficiency*	1,0	4	4,0
T_4	*Complex internal proccesing*	1,0	4	4,0
T_5	*Reusability of code*	1,0	2	2,0
T_6	*Easy to install*	0,5	5	2,5
T_7	*Ease of use*	0,5	3	1,5
T_8	*Portability*	2,0	1	2,0
T_9	*Ease of change*	1,0	3	3,0
T_{10}	*Concurrency*	1,0	2	2,0
T_{11}	*Special security objectives included*	1,0	2	2,0
T_{12}	*Direct access for third parties*	1,0	5	5,0
T_{13}	*Special user training required*	1,0	3	3,0
Technical Factor Value (TFactor)				45,0

Rumus yang digunakan untuk mendapatkan nilai *Technical Complexity Factor*:

$$TCF = 0{,}6 + (0{,}01 * TFactor)$$
$$TCF = 0{,}6 + (0{,}01 * 45)$$
$$TCF = 1{,}05$$

5. Menghitung nilai faktor lingkungan untuk memperoleh nilai *Environmental Factor* (EF)

Tabel 9. Environmental Factor Table

Factor Number	*Description*	*Weight*	*Assigned Value (0–5)*	*Weighted Value*
E_1	*Familiarity with system development process being used*	1,5	4	6,0
E_2	*Application experience*	0,5	3	1,5
E_3	*Object-oriented experience*	1,0	4	4,0
E_4	*Lead analyst capability*	0,5	4	2,0
E_5	*Motivation*	1,0	3	3,0
E_6	*Requirements stability*	2,0	4	8,0
E_7	*Part time staff*	-1,0	0	0,0
E_8	*Difficulty of programming language*	-1,0	3	-3,0
Environmental Factor Value (EFactor)				21,5

Rumus yang digunakan untuk mendapatkan nilai *Environmental Factor*:

$$EF = 1{,}4 + (-0{,}03 * EFactor)$$
$$EF = 1{,}4 + (-0{,}03 * 21{,}5)$$
$$EF = 0{,}76$$

6. Nilai UUCP, TCF dan EF digunakan untuk menghitung nilai *Unadjusted Case Points* dengan rumus:
   $UCP = UUCP * TCF * EF$
   $UCP = 149 * 1{,}05 * 0{,}76$
   $UCP = 118{,}90$
7. Berdasarkan data di Tabel 9, dapat diketahui nilai *Person Hour Multiplier* (PHM) adalah 20.
8. *Effort* aplikasi didapat dengan menggunakan rumus:

$Effort = UCP * PHM$
$Effort = 118{,}90 * 20$
$Effort = 2.378$

Dari hasil penghitungan estimasi proyek perangkat lunak sistem informasi depot dengan menggunakan metode *use case points* diperoleh hasil *effort* adalah sebesar 2.378 jam. Hasil pengujian semua proyek yang ada dengan metode *use case points* dapat dilihat pada Gambar 10.

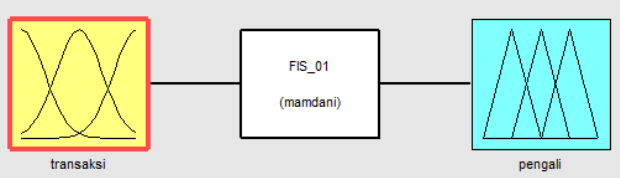
No	Nama Proyek	UAW	UUCW	UUCP	TFactor	Efactor	UCP	PHM	Effort
1	PROYEK 01 - SISTEM INFORMASI SEKOLAH	12.00	140.00	152.00	1.17	0.85	151.16	20	3023
2	PROYEK 02 - SISTEM INFORMASI DEPOT	14.00	135.00	149.00	1.05	0.76	118.90	20	2378
3	PROYEK 03 - SISTEM INFORMASI BAYPLAN	5.00	80.00	85.00	1.14	0.82	79.46	20	1589
4	PROYEK 04 - SISTEM INFORMASI PENERIMAAN MAHASISWA BARU	9.00	65.00	74.00	1.16	0.76	65.24	20	1305
5	PROYEK 05 - SISTEM INFORMASI AKADEMIK	12.00	135.00	147.00	1.09	0.85	136.20	20	2724
6	PROYEK 06 - SISTEM INFORMASI PERPUSTAKAAN	3.00	50.00	53.00	1.16	0.85	52.26	20	1045

Gambar 10. Hasil Pengujian dengan Metode Use Case Points

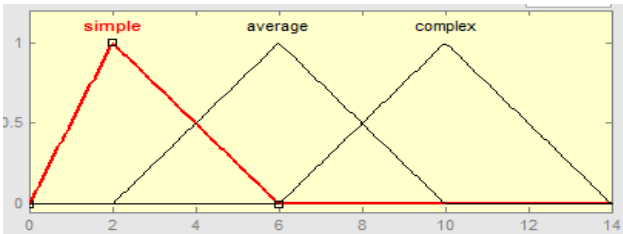
6.1.2 Metode *Fuzzy Use Case Points*

Metode *fuzzy use case points* yang digunakan dalam penelitian ini merupakan modifikasi metode *use case points* dengan menggunakan logika *fuzzy* untuk memberikan nilai klasifikasi *use case*.

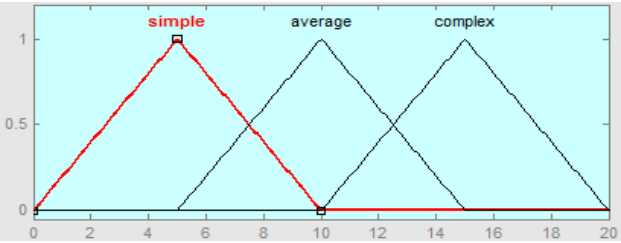
Logika *fuzzy* dibuat menggunakan *fuzzy inference system* metode Mamdani, terdiri dari variabel masukan dan keluaran yang masing-masing terdiri dari tiga kurva segitiga.



Gambar 11. FIS Metode Mamdani



Gambar 12. Variabel Masukan



Gambar 13. Variabel Keluaran

Logika *fuzzy* yang dibuat juga memerlukan pengaturan rules sebagai berikut:
If transaksi *is simple then* pengali *is simple*
If transaksi *is average then* pengali *is average*
If transaksi *is complex then* pengali *is complex*

Dari hasil pengujian logika dengan memberikan nilai masukan berupa jumlah transaksi akan menghasilkan nilai pengali sesuai Tabel 10.

Tabel 10. Hasil Keluaran Logika Fuzzy

Nilai masukan (transaksi)	Nilai keluaran (pengali)
1	5,00
2	5,00
3	6,45
4	7,50
5	8,55
6	10,00
7	11,40
8	12,50
9	13,60
10	15,00
11	15,00
12	15,00

6.2 Evaluasi dan Validasi Hasil

Dengan menggunakan data dari aplikasi yang sama, dilakukan penelitian untuk menghitung *effort* dengan menggunakan logika *fuzzy*. Tahapan yang dilakukan untuk menghitung *effort* dengan menggunakan logika *fuzzy* tidak banyak berbeda dengan metode *use case points*, yang membedakan hanya ketika mencari nilai *anadjusted use case weight*. Untuk menentukan nilai *unadjusted use case weight* menggunakan Tabel 10.

Nilai *UAW*, *TCF*, *EF* dan *PHM* menggunakan nilai yang diperoleh dengan metode *use case points*. Sedangkan nilai *UUCW* untuk aplikasi sistem informasi depot apabila menggunakan logika *fuzzy* tercantum pada Tabel 11.

Tabel 11. Klasifikasi *use case* dengan logika *fuzzy*

No	*Use case*	Trans aksi	Nilai
1	Melakukan *login*	1	5,00
2	Memasukkan *password* dan level akses	1	5,00
3	Mengatur *general setting*	2	5,00
4	*Input* pergerakan *in* depot	3	6,45
5	*Input* pergerakan *out* depot	5	8,55
6	*Input* kondisi *contr*	2	5,00
7	*Input* kondisi DM	8	12,50
8	*Input* kondisi AV	2	5,00
9	Melakukan transaksi pembayaran	8	12,50
10	Membuat laporan 1	10	15,00
11	Membuat laporan 2	10	15,00
12	Melakukan konfigurasi data master	10	15,00
13	Menggunakan *tools*	5	8,55
14	Membuat tagihan	7	11,40
Nilai *unadjusted use case weight*			129,95

Dari hasil pengujian diperoleh nilai *UUCW* adalah 129,95. Dengan data yang ada, nilai *UAW* adalah 14, sehingga nilai *UUCP* menjadi 143,95 dan nilai *UCP* 114,87.

Effort yang didapat dengan metode *fuzzy use case points* adalah 2.297 jam. Hasil lengkap pengujian dengan metode *fuzzy use case points* dapat dilihat pada Gambar 14.

No	Nama Proyek	UAW	FUUCW	FUUCP	TFactor	Efactor	FUCP	PHM	Effort FUCP
1	PROYEK 01 - SISTEM INFORMASI SEKOLAH	12.00	123.90	135.90	1.17	0.85	135.15	20	2703
2	PROYEK 02 - SISTEM INFORMASI DEPOT	14.00	129.95	143.95	1.05	0.76	114.87	20	2297
3	PROYEK 03 - SISTEM INFORMASI BAYPLAN	5.00	67.50	72.50	1.14	0.82	67.77	20	1355
4	PROYEK 04 - SISTEM INFORMASI PENERIMAAN MAHASISWA BARU	9.00	63.60	72.60	1.16	0.76	64.00	20	1280
5	PROYEK 05 - SISTEM INFORMASI AKADEMIK	12.00	123.60	135.60	1.09	0.85	125.63	20	2513
6	PROYEK 06 - SISTEM INFORMASI PERPUSTAKAAN	3.00	47.05	50.05	1.16	0.85	49.35	20	987

Gambar 14. Hasil Pengujian dengan Metode *Fuzzy Use Case Points*

6.3 Pembahasan

Semua data *use case metrics* dari enam aplikasi dihitung dengan menggunakan metode *use case points* dan *fuzzy use case points* sehingga diperoleh data *effort* untuk setiap aplikasi. *Effort* sesungguhnya dibandingkan dengan *effort* dari metode *use case points* dan *fuzzy use case points* menggunakan rumus membandingkan nilai *effort* yaitu MMRE (Briand, 1998) dan MMER (Kitchenham, 2001) serta dilengkai pula dengan rumus *Mean Error with Standard Deviation* (Nassif, 2010).

Hasil perhitungan *effort* dari enam proyek perangkat lunak terlihat pada Gambar 15.

No	Nama Proyek	Effort			MER		MRE		Error	
		Actual	UCP	FUCP	UCP	FUCP	UCP	FUCP	UCP	FUCP
1	PROYEK 01 - SISTEM INFORMASI SEKOLAH	2200	3023	2703	0.27	0.19	0.37	0.23	823	503
2	PROYEK 02 - SISTEM INFORMASI DEPOT	2000	2378	2297	0.16	0.13	0.19	0.15	378	297
3	PROYEK 03 - SISTEM INFORMASI BAYPLAN	1000	1589	1355	0.37	0.26	0.59	0.36	589	355
4	PROYEK 04 - SISTEM INFORMASI PENERIMAAN MAHASISWA BARU	1200	1305	1280	0.08	0.06	0.09	0.07	105	80
5	PROYEK 05 - SISTEM INFORMASI AKADEMIK	2400	2724	2513	0.12	0.04	0.13	0.05	324	113
6	PROYEK 06 - SISTEM INFORMASI PERPUSTAKAAN	900	1045	987	0.14	0.09	0.16	0.10	145	87
Rerata					0.19	0.13	0.26	0.16	394.00	239.17
Standar Deviasi									249.25	158.52
Improvement					+6%		+10%			

Gambar 15. Perbandingan Hasil Pengujian dengan Dua Metode

Dari hasil pengujian menghitung estimasi proyek perangkat lunak antara metode *use case point* dan *fuzzy use case point*, ditemukan perbedaan hasil yang mempengaruhi akurasi. Peningkatan akurasi mendekati *effort* aktual menggunakan *fuzzy use case points* sebesar 6% untuk MMER dan 10% bila menggunakan metode pengukuran MMRE. Rerata standar deviasi kesalahan metode *use case points* sebesar 394,00 $\pm$ 249,25 sedangkan metode *fuzzy use case points* sebesar 239,17 $\pm$ 158,52. Nilai rerata standar deviasi kesalahan metode *fuzzy use case points* lebih kecil bila dibandingkan dengan nilai metode *use case points*. Dengan demikian, dari tiga metode pengukuran dapat terlihat bila menghitung estimasi dengan logika *fuzzy* akan lebih mendekati *effort* aktual.

6.4 Tools

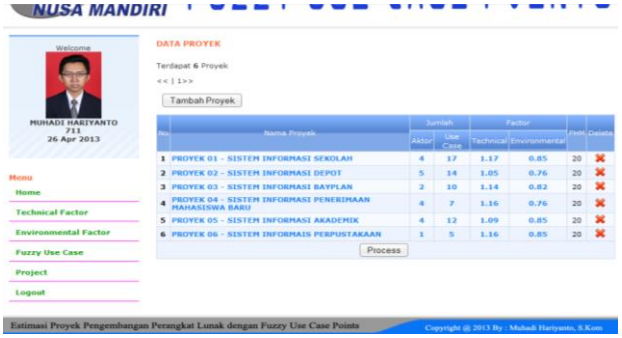
Dalam pengolahan data pada penelitian ini, penulis merancang sebuah aplikasi berbasis *web* dengan menggunakan bahasa pemrograman PHP sebagai alat bantu untuk mempermudah dalam proses perhitungan *effort* yang menggunakan metode *use case points* dan *fuzzy use case points*.

Aplikasi ini hanya digunakan untuk menyimpan data *use case metrics* dari enam proyek perangkat lunak kemudian

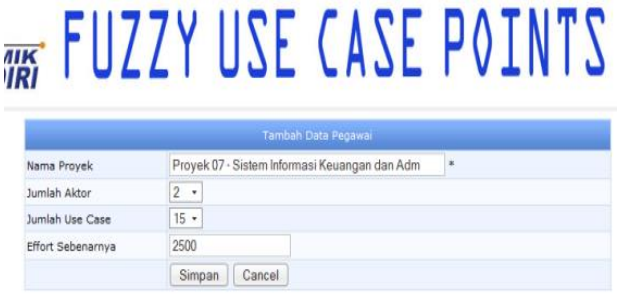
diolah dengan rumus dari metode *use case points* dan *fuzzy use case points*, Agar aplikasi dapat menghitung dengan metode *fuzzy use case points*, maka hasil pengolahan *fuzzy inference system* di masukkan ke dalam sistem.

Gambar 16. Tampilan Awal Aplikasi

Gambar 17. Halaman *Fuzzy Use Case Points*



Gambar 18. Halaman Project



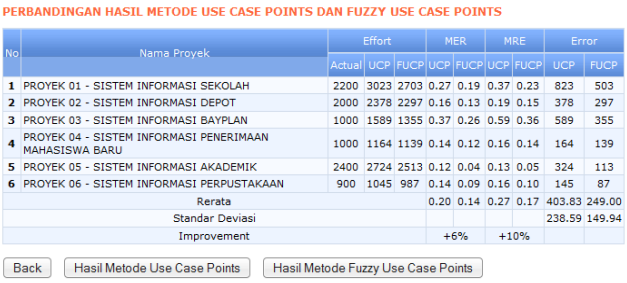
Gambar 19. Halaman Menambahkan Proyek Baru

Gambar 20. Halaman Menambah dan Mengubah Data Aktor

Gambar 21. Halam Edit Nama dan Jumlah Transaksi Setiap *Use Case*

Gambar 22. Halaman Untuk Mengisi Nilai *Technical Factor*

Gambar 23. Halaman Untuk Mengisi Nilai *Environmental Factor*



PERBANDINGAN HASIL METODE USE CASE POINTS DAN FUZZY USE CASE POINTS

No	Nama Proyek	Effort			MER		MRE		Error	
		Actual	UCP	FUCP	UCP	FUCP	UCP	FUCP	UCP	FUCP
1	PROYEK 01 - SISTEM INFORMASI SEKOLAH	2200	3023	2703	0.27	0.19	0.37	0.23	823	503
2	PROYEK 02 - SISTEM INFORMASI DEPOT	2000	2378	2297	0.16	0.13	0.19	0.15	378	297
3	PROYEK 03 - SISTEM INFORMASI BAYPLAN	1000	1589	1355	0.37	0.26	0.59	0.36	589	355
4	PROYEK 04 - SISTEM INFORMASI PENERIMAAN MAHASISWA BARU	1000	1164	1139	0.14	0.12	0.16	0.14	164	139
5	PROYEK 05 - SISTEM INFORMASI AKADEMIK	2400	2724	2513	0.12	0.04	0.13	0.05	324	113
6	PROYEK 06 - SISTEM INFORMASI PERPUSTAKAAN	900	1045	987	0.14	0.09	0.16	0.10	145	87
	Rerata				0.20	0.14	0.27	0.17	403.83	249.00
	Standar Deviasi								238.59	149.94
	Improvement				+6%		+10%			

Back | Hasil Metode Use Case Points | Hasil Metode Fuzzy Use Case Points

Gambar 24. Halaman Hasil Proses *Use Case Metrics*

7 KESIMPULAN

Menghitung estimasi proyek pengembangan perangkat lunak dengan menggunakan metode *fuzzy use case points* memberikan kenaikan akurasi mendekati *effort* sebenarnya bila dibandingkan dengan menggunakan *use case points*. Kenaikan akurasi sebesar 6% bila menggunakan metode perhitungan MMER dan 10% dengan metode MMRE. Nilai standar deviasi kesalahan juga lebih kecil metode *fuzzy use case points* yaitu sebesar 239,17 $\pm$ 158,52. Sedangkan nilai standar deviasi kesalahan metode *use case points* sebesar 394,00 $\pm$ 249,25. Sehingga dapat disimpulkan bahwa estimasi menggunakan metode *fuzzy use case points* lebih mendekati *effort* sebenarnya dalam proses menghitung estimasi pengembangan perangkat lunak.

DAFTAR REFERENSI

Anda, Bente., (2002). Comparing Effort Estimates Based on Use Case Points with Expert Estimates. *Empirical Assessment in Software Engineering (EASE 2002).*

Away, Gunaidi Abdia,.(2010). *The Shortcut of Matlab Programming*. Bandung: Informatika.

Albrecht, Allan J., (1979). Measuring Application Development Productivity. *Joint SHARE/GUIDE/IBM Appplication Development Symposium*.

Berndtssom, M., Hansson, J., Olsson, N., & Lundell, B. (2008). *A Guide for Students in Computer Science and Information Systems* (2nd ed). London: Springer.

Briand, Lionel C., Emam, Khaled El., Surmann, Dagmar., Wieczorek, Isabella., & Maxwell, Katrina D., (1998). An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques. *International Software Engineering Research Network Technical Report ISERN-98-27*.

Cox, Earl., (1994). *The Fuzzy System Handbook (A Practitioner's Guide to Building, Using, and Maintaining Fuzzy Systems)*. Massachusetts: Academic Press, Inc.

Dennis, A., Wixom, B. H., & Tegarden, D., (2005). *System Analysis and Design with UML Version 2.0 An Object-Oriented Approach* (2nd ed). USA: John Wiley & Sons, Inc.

Dawson, C. W. (2009). *Projects in Computing and Information System A Student's Guide* (2nd ed). England: Addison-Wesley

Fan, Wang., Xiahou, Yang., Xiachun, Zhu., & Lu, Chen. (2009). Extended Use Case Points Method for Software Cost Estimation. *The Center of Technology and Business Innovation.*

Gray, D.E. (2004). *Doing Research in the Real World.* India: SAGE

Gustafson, David A. (2002). *Theory and Problems of Software Engineering*. USA: McGraw-Hill.

Iraji, Mohammad Saber., Aboutalebi, Majid., & Motameni, Homayun., (2012). Effort Estimate with Neuro Fuzzy Use Case Point Based on Exact Weights. *Progress in Computing Applications Volume 1 Number 1, March 2012*.

Langer, Arthur M. (2008). *Analysis and Design of Information Systems* (3rd ed). London: Springer

Lee, Kwang H, (2005). *First Course on Fuzzy Theory and Applications*. German: Springer.

Lily, John H., (2010). *Fuzzy Control and Identification.* New Jersey: John Wiley

Lughofer, Edwin. (2011). *Envolving Fuzzy Systems, Methodologies, Advanced Concepts and Applications*. German: Springer.

Kamal, Mohammed Wajahat., & Ahmed, Moataz A., (2011). A Proposed Framework for Use Case bassed Effort Estimation using Fuzzy Logic: Building upon the outcomes of a Systematic Literature Review. *International Journal on New Computer Architectures and Their Applications (IJNCAA) 1(4):976-999 The Society of Digital Information and Wireless Communications, 2011 (ISSN: 2220-9085).*

Karner, Gustav. (1993). Resource Estimation for Objectory Projects. *Objective Systems SF AB, Rational Software.*

Kitchenham, B.A., Pickard, L.M., MacDonell, S.G., & Shepperd, M.J., (2001). What accuracy statistics really measure. *IEE Proc-Softw., Vol. 148, No. 3, June 2001*.

Kothari, C. R. (2004). *Research Methodology Methodes and Technique* (2nd ed). India: New Age International

Kusumadewi, S., & Purnomo, H., (2010). *Aplikasi Logika Fuzzy untuk Pendukung Keputusan* (2nd ed). Yogyakarta: Graha Ilmu

Nassif, Ali Bou., Capretz, Luiz Fernaando., & Ho, Danny., (2010). Enhancing Use Case Points Estimation Method Using Soft Computing Techniques. *Journal of Global Research in Computer Science Volume 1, No. 4, November 2010*

Sivanandam, S.N., Sumathi, S., & Deepa, S.N., (2007). *Introduction to Fuzzy Logic using MATLAB*. German: Springer.

Sommerville, Ian. (2007). *Software Engineering* (8th ed). England: Addison-Wesley

Webopedia Computer Dictionary. (n.d). October 10, 2012. http://www.webopedia.com/TERM/F/fuzzy_logic.html

Yeates, D., & Wakefield, T. (2004). *System Analysis and Design*. England: Pearson Education Limited.

BIOGRAPHY OF AUTHORS

Muhadi Hariyanto. Menyelesaikan pendididikan S1 dan S2 di STMIK Nusa Mandiri, Jakarta. Saat ini bekerja sebagai programmer di Universitas Tarumanagara dan sebagai dosen S1 di STMIK Nusa Mandiri. Research interest di bidang software engineering.

Romi Satria Wahono. Menempuh pendidikan S1 (B.Eng), S2 (M.Eng) di bidang Software Engineering di Department of Computer Science di Saitama University, Jepang. Menyelesaikan pendidikan S3 (Ph.D) bidang Software Engineering di Universiti Teknikal Malaysia Melaka. Founder dan CEO PT Brainmatics, IlmuKomputer.Com dan Intelligent Systems Research Center. Mengajar di beberapa program pasca sarjana ilmu komputer di Indonesia, salah satunya di Universitas Dian Nuswantoro. Research interest di bidang Software Engineering dan Machine Learning. Professional member dari IEEE CS, ACM dan PMI.

Penerapan Java Dynamic Compilation pada Metode Java Customized Class Loader untuk Memperbaharui Perangkat Lunak pada saat Runtime

Tory Ariyanto, Romi Satria Wahono and Purwanto
Fakultas Ilmu Komputer, Universitas Dian Nuswantoro
toryaquino@gmail.com, romi@romisatriawahono.net, purwanto@dsn.dinus.ac.id

Abstract: Proses pembaharuan perangkat lunak diperlukan untuk menjaga kehandalan sebuah perangkat lunak agar bisa berjalan dengan baik. Sebuah perangkat lunak yang memiliki tingkat operasional tinggi sehingga tidak diperbolehkan untuk melakukan restart, memerlukan sebuah metode dimana metode tersebut dapat melakukan proses pembaharuan dengan cepat tanpa melakukan restart. Untuk mengatasi hal tersebut dapat digunakan metode *Java Customized Class Loader* (JCCL). Penerapan JCCL untuk melakukan pembaharuan perangkat lunak dengan tidak memperbolehkan restart memiliki kendala yaitu lambatnya proses pembaharuan. Pada penelitian ini, akan diterapkan *Java Dynamic Compilation* (JDC) untuk meningkatkan efisiensi waktu metode JCCL dalam melakukan pembaharuan perangkat lunak tanpa melakukan restart (JCCL+JDC). Untuk menguji coba metode yang diusulkan, digunakan aplikasi permainan *snake game*, aplikasi permainan ini telah digunakan juga pada penelitian sebelumnya. Berdasarkan hasil eksperimen, metode yang diusulkan dalam penelitian ini (JCCL+JDC) memiliki waktu yang lebih efisien dari metode JCCL sebelum dioptimalkan dengan metode JDC. Waktu yang dihasilkan dari metode yang diusulkan menyesuaikan kompleksitas algoritma dari setiap *method* yang ditambahkan dalam proses pembaharuan.

Keywords: Pembaharuan Perangkat Lunak, *Java Customized Class Loader, Java Dynamic Compilation*

1 PENDAHULUAN

Pembaharuan perangkat lunak pada saat *runtime* atau yang disebut dengan *Dynamic Software Update* (DSU) adalah teknik melakukan pembaharuan perangkat lunak ketika perangkat lunak tersebut sedang berjalan tanpa menimbulkan waktu jeda, oleh karena itu efisiensi waktu atau mengurangi waktu tunggu dari proses pembaharuan perangkat lunak merupakan hal yang diperlukan (Seifzadeh, Abolhassani, & Moshkenani, 2012). DSU juga disebut sebagai *on-the-fly program modification*, *online version change*, *hot-swap* dan *dynamic software maintenance* (Orso, Rao, & Harrold, 2002).

Pembaharuan perangkat lunak pada umumnya dilakukan untuk memperbaiki kesalahan kode program dan prosedur yang terdapat pada sebuah perangkat lunak. Ketika memperbaharui perangkat lunak, pada umumnya harus melakukan *restart* pada perangkat lunak yang telah diperbaharui sebelum melihat hasil pembaharuannya. Hal seperti ini mengurangi nilai efisiensi bahkan mengganggu (Kim, Tilevich, & J, 2010). Setelah memodifikasi dan mengkompilasi kode program yang baru, akan lebih baik jika hasilnya dapat secara langsung dimasukkan ke dalam perangkat lunak yang sedang berjalan tanpa perlu menghentikan perangkat lunak atau menunggu proses yang sedang berjalan selesai. Sebagai contoh memodifikasi *event* yang berada dalam sebuah tombol, tidak perlu menutup seluruh perangkat lunak yang sedang berjalan atau menunggu proses lain selesai (Würthinger, Wimmer, & Stadler, 2013).

Untuk menangani permasalahan pembaharuan perangkat lunak secara dinamis dimana tidak memperbolehkan komputer untuk *restart* dan mengharuskan proses pembaharuan yang efisien agar tidak terjadi jeda, terdapat beberapa pendekatan atau metode. Metode tersebut adalah *Customized Java Virtual Machine* (CJVM), *Java Wrapper* (JW), Jav Adaptor (JA) dan *Java Customized Class Loader* (JCCL).

Menurut Pukall (Pukall et al., 2011) CJVM dapat menyelesaikan permasalahan mengenai pembaharuan perangkat lunak pada saat *runtime*, akan tetapi metode ini tergantung dengan versi dari *Java Virtual Machine* (JVM). Hal ini menyebabkan metode CJVM tidak fleksibel untuk diterapkan pada seluruh versi JVM. Masalah rendahnya tingkat fleksibilitas metode CJVM dapat diatasi oleh metode JW, metode ini tidak tergantung pada JVM versi tertentu sehingga lebih fleksibel. Terlepas dari kemampuan JW yang fleksibel, JW memiliki permasalahan pada kemampuan untuk memperbaharui perangkat lunak. JW hanya bisa mengganti isi dari *method* pada sebuah kelas Java. Dikarenakan metode JW hanya bisa mengganti isi *method*, maka kekurangan ini dicoba diatasi oleh metode JA. Untuk mengatasi masalah yang terdapat pada JW diusulkan metode JA dimana metode ini dapat mengatasi permasalahan terbatasnya kemampuan melakukan pembaharuan yang dimiliki JW. Metode JA dapat mengganti seluruh bagian dari kode program serta fleksibel untuk dapat diterapkan pada seluruh versi JVM. Walaupun JA fleksibel terhadap seluruh versi JVM, akan tetapi efisiensi proses pembaharuan dari metode ini bukan menjadi tujuan utama,tujuan utama dari metode ini adalah agar bisa mengganti seluruh bagian kode program dan tidak tergantung terhadap JVM versi tertentu. Karena keterbatasan efisiensi proses pembaharuan, maka meningkatkan efisiensi proses pembaharuan menjadi tujuan lebih lanjut dari penelitian tentang JA. Permasalahan perlunya mengurangi waktu pada proses pembaharuan perangkat lunak juga terdapat pada penelitian Thomas Wuthingler (Würthinger et al., 2013), dalam penelitiannya dilakukan pendekatan dengan metode CJVM dimana dalam penelitian tersebut mengurangi waktu eksekusi dari kode program yang lama, merupakan langkah yang dapat dilakukan dalam penelitian berikutnya.

Metode-metode di atas memiliki kode sumber yang tertutup sehingga tidak bisa diteliti oleh umum untuk dikembangkan lebih lanjut. Melihat hal tersebut para peneliti DSU dan perangkat lunak yang memiliki waktu operasional tinggi lebih banyak yang menggunakan metode JCCL, metode JCCL dapat berjalan pada seluruh versi JVM. Metode JCCL melakukan pembaharuan dengan memodifikasi C*lass Loader* yang dimiliki oleh Java. *Class Loader* merupakan kelas dalam Java yang digunakan untuk memuat dan mengatur kelas yang sedang berjalan pada JVM secara dinamis, teknik ini secara

umum banyak digunakan dalam melakukan pembaharuan perangkat lunak yang sedang berjalan (Zhang, 2007). Kelas ini dapat dimodifikasi untuk dapat melakukan proses pemuatan ulang kelas kedalam JVM secara dinamis. Metode ini telah digunakan oleh OSGi Service Platform, Oracles FastSwap dalam memperbaharui komponennya yang sedang berjalan di JVM serta Javeleon yang memungkinkan pembaharuan secara fleksibel dan netBeans yang berbasis perangkat lunak juga telah menerapkan metode ini (Pukall et al., 2011). Metode ini memiliki kode sumber terbuka untuk umum, sehingga JCCL dapat dikembangkan lagi agar waktu proses pembaharuan lebih cepat jika jumlah *method* yang terdapat pada perangkat lunak terus bertambah (Pukall et al., 2011).

Walaupun metode JCCL dapat berjalan pada seluruh versi JVM akan tetapi metode ini mengalami penurunan efisiensi seiring dengan bertambahnya jumlah *method* pada kelas yang akan diperbaharui (Pukall et al., 2011). Dalam penelitian Zhang tentang *Dynamic Update Transactions*, digunakan metode JCCL yang dikombinasikan dengan metode *Java Reflection* untuk melakukan proses pembaharuan perangkat lunak secara dinamis pada seluruh versi JVM (Zhang, 2007). Pada penelitian tersebut proses pembaharuan dilakukan terhadap seluruh bagian program ketika *runtime* sehingga membutuhkan proseses pembaharuan yang besar pada program yang sedang berjalan sehingga berdampak pada berkurangnya efisiensi proses pembaharuan (Pukall et al., 2011).

Untuk meningkatkan efisiensi eksekusi kode program, terdapat metode yang bisa digunakan, yaitu *Java Dynamic Compilation* (JDC). JDC adalah metode yang telah digunakan oleh Sun hotSpot dan IBM's Jalapeno untuk mengoptimasi serta meningkatkan efisiensi eksekusi kode Java pada saat *runtime* sehingga waktu eksekusi program menjadi lebih efsien (Hayden, Smith, & Foster, 2012). JDC meningkatkan kinerja JVM dengan cara menjalankan kode atau *method* Java yang telah dikompilasi akan tetapi belum dimuat ke dalam JVM sebelumnya (Fong, 2012). Berdasarkan hal tersebut, dalam penelitian ini akan digunakan metode JDC untuk meningkatkan efisiensi JCCL dalam melakukan proses pembaharuan perangkat lunak pada saat *runtime*.

2 PENELITIAN TERKAIT

Pada penelitian Zhang (Zhang, 2007), dilakukan pendekatan dengan menggunakan metode JCCL berbasis *Java Reflection*. Penelitian ini menyelesaikan permasalahan dari penelitian sebelumnya, dimana proses pembaharuan hanya terbatas memperbaharui *method* pada kelas yang sama dan harus mengubah versi dari JVM sehingga harus melakukan modifikasi terhadap setiap versi JVM. Metode ini menggantikan *Class Loader* Java, Sehingga perangkat lunak yang akan melakukan proses pembaharuan pada saat *runtime* harus menerapkan *Class Loader* yang telah dimodifikasi agar bisa melakukan pembaharuan pada seluruh Kelas dan *method* yang ada tanpa melakukan perubahan struktur atau versi dari JVM. Terdapat dua tahap dalam menggunakan metode JCCL, tahap pertama adalah memanggil objek menggunakan *Class Loader* sendiri dan tahap kedua mengubah *method* dengan menggunakan *Java Reflection*.

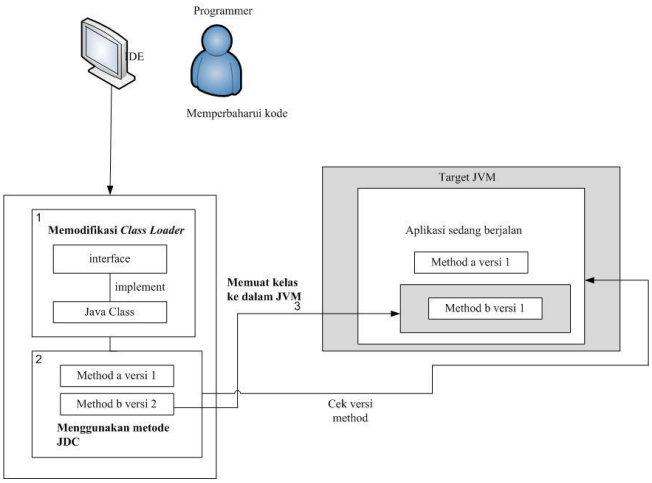
Pendekatan pembaharuan perangkat lunak menggunakan JCCL juga dilakukan dalam penelitiannya Kai (Gregersen & Koskimies, 2012). Penelitian ini memfokuskan pembuatan sebuah *tool* yang diintegrasikan ke dalam *framework* tertentu, sehingga perlu modifikasi ulang apabila terdapat *framework* baru. Dalam penelitian tersebut, memungkinkan beberapa versi dari objek untuk secara bersama berada dalam sistem yang sedang berjalan. Hal ini dapat terjadi dengan cara menggunakan Java Bootstrap atau disebut juga dengan (Javeleon) yang menciptakan *Class Loader* baru untuk setiap versi baru dari perangkat lunak yang akan diperbaharui. Cara keja dari metode yang diusulkan ini adalah dengan mengubah objek yang sudah berjalan di dalam JVM kedalam versi yang lebih baru.

Perbedaan yang terjadi pada metode JCCL berbasis Java Reflection dan metode JCCL berbasi Java Boostrap (Javeleon) adalah pada metode pembaharuan yang dilakukan, metode yang pertama melakukan proses pembaharuan dengan memuat ulang seluruh *method* sedangkan metode yang kedua melakukan proses pembaharuan dengan mengubah objek yang sudah berjalan pada JVM kedalam versi yang lebih baru.

3 METODE YANG DIUSULKAN

Class Loader standar yang dimiliki Java merupakan kelas yang terdapat pada Java untuk digunakan sebagai kelas yang bertugas memuat kelas dari program yang dibuat ke dalam JVM. Kelas ini dapat dimodifikasi dan mengontrol apa saja yang harus dilakukan oleh JVM. Dalam penelitian ini, *Class Loader* standar yang dimiliki Java akan dimodifikasi dengan menambahkan metode JDC sehingga dapat digunakan untuk memperbaharui perangkat lunak pada saat *runtime* dengan efisien. Proses pembaharuan perangkat lunak akan menjadi efisien karena metode JDC akan memeriksa *method* mana yang diperbaharui, sehingga hanya *method* itulah yang akan dimuat ulang ke dalam JVM.

Metode yang diusulkan dalam penelitian ini adalah JCCL berbasis JDC seperti pada Gambar 1.



Gambar 1 Metode yang Diusulkan

Perbedaan dengan metode sebelumnya yang menggunakan pendekatan JCCL terletak pada teknik pembaharuan kelas. Metode yang diusulkan yaitu JCCL berbasis JDC hanya memuat ulang *method* yang diperbaharui saja, *method* lama yang tidak diperbaharui dibiarkan tetap berjalan pada JVM tanpa dimuat ulang seperti pada Tabel 1.

Tabel 1 Perbedaan Dengan Penelitian Sebelumnya

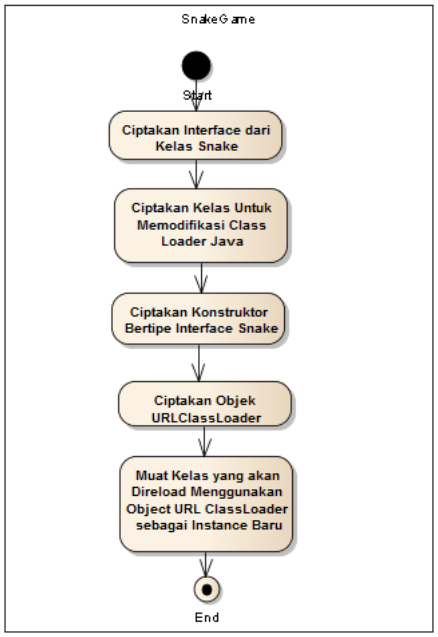
Penelitian	Metode	Perbedaan Dengan Penelitian Lainnya
Java Customized Class Loader berbasis Java Reflection (S. Z. S. Zhang dan L. H. L. Huang, 2007)	JCCL berbasis Java Reflection	Pembaharuan dilakukan dengan memuat ulang seluruh *method* yang terdapat pada JVM
Java Customized Class Loader berbasis Java Bootstrap (Gregersen dan Allan Raundahl, 2012) Koskimies dan Kai, 2012	JCCL berbasis Java boostrap	Pembaharuan dilakukan dengan mengubah objek yang terdapat pada JVM kedalam versi yang baru
Metode yang diusulkan (JCCL+JDC)	JCCL berbasis JDC	Pembaharuan dilakukan hanya dengan memuat ulang *method* yang diperbaharui saja, *method* lama yang tidak diperbaharui dibiarkan tetap berjalan pada JVM tanpa dimuat ulang

Metode yang diusulkan dalam penelitian ini memiliki langkah-langkah pembaharuan sebagai berikut:

1. Memodifikasi *Class Loader*
2. Menggunakan Metode JDC
3. Memuat Kelas ke dalam JVM

1. Memodifikasi Class Loader

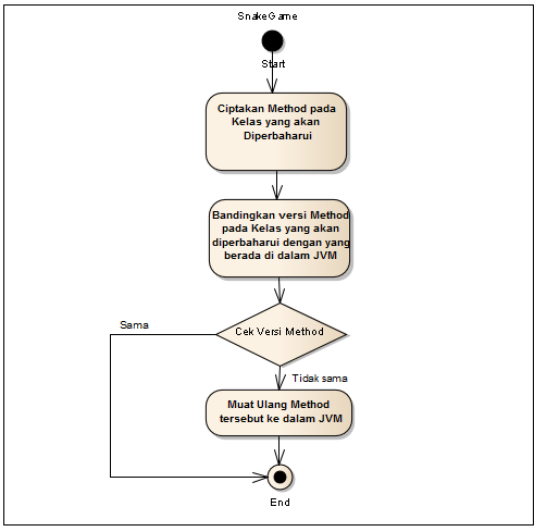
Dalam penelitian ini, *Class Loader* standar yang dimiliki Java perlu dimodifikasi agar dapat digunakan untuk memuat kelas kedalam JVM pada saat *runtime* dengan waktu seminimal mungkin. Modifikasi yang dilakukan memanfaatkan kelas yang terdapat pada Java yaitu *URL Class Loader*. Adapun cara penggunaan *Class Loader* standar yang dimiliki Java dan *URL Class Loader* adalah dengan membuatkan *interface* pada kelas yang akan diperbaharui dan selanjutnya *interface* tersebut akan digunakan oleh *Class Loader* yang dimodifikasi sebagai tipe konstruktor. *URL Class Loader* bertugas memanggil kelas yang akan dimuat ke dalam JVM. Setelah menciptakan objek *URL Class Loader*, selanjutnya objek tersebut bertugas memuat ulang kelas dari program yang akan diperbaharui ke dalam JVM menggunakan *method loadClass* dan menciptakan *instance* baru Gambar 2.



Gambar 2 Diagram Aktifitas Memodifikasi *ClassLoader*

2. Menggunakan Metode JDC

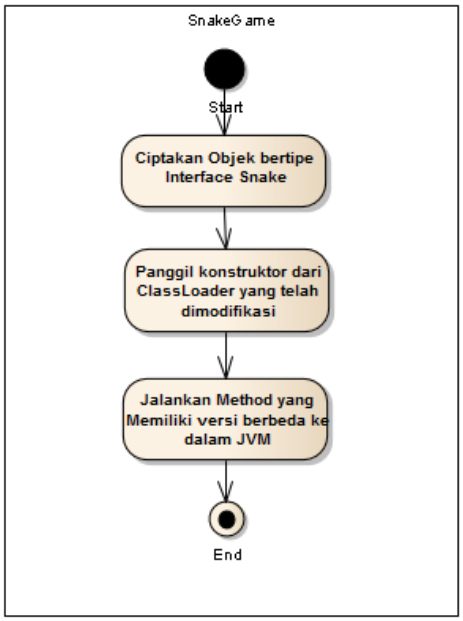
Dalam menggunakan metode JDC, kelas yang akan dimuat ke dalam JVM dibuatkan versi pada setiap *method*. Hal tersebut berfungsi, agar *method* yang tidak memiliki perubahan tidak dijalankan di dalam JVM. *Method* yang akan dijalankan di dalam JVM hanya *method* baru yang sebelumnya belum pernah dimuat ke dalam JVM. Di dalam Gambar 3 terdapat proses untuk mengecek apakah versi *method* yang terdapat di dalam kelas utama yang sudah dimuat di dalam JVM sama dengan versi *method* yang terdapat pada kelas yang akan diperbaharui. Jika sama maka *method* yang terdapat di dalam kelas yang akan diperbaharui tersebut dianggap sebagai *method* lama dan tidak akan dimuat ulang ke dalam JVM, jika versi *method* tersebut berbeda dengan yang ada di dalam JVM maka *method* tersebut dianggap sebagai *method* baru dan akan dimuat ulang ke dalam JVM.



Gambar 3 Diagram Aktifitas Penggunaan Metode JDC

3. Memuat Kelas ke Dalam JVM

Setelah kelas yang akan diperbaharui memiliki *method* dengan kode untuk mengecek versi, langkah terakhir adalah memuat kelas yang akan diperbaharui ke dalam JVM dengan menciptakan objek dari kelas yang akan diperbaharui meggunakan JCCL yang telah dimodifikasi dan ditambahakan metode JDC seperti pada Gambar 4.



Gambar 4 Diagram Aktifitas Memuat Kelas ke dalam JVM

4 HASIL PENELITIAN

Untuk mengetahui efisiensi waktu dari metode yang diusulkan, maka dilakukan ujicoba terhadap metode JCCL sebelum dioptimalkan dengan JDC dan sesudah dioptimalkan dengan JDC mengunakan data uji *snake game.* Perangkat lunak ini didapat dari situs pengembang perangkat lunak Java

Proses pengujian dilakukan sebanyak 30 kali percobaan menggunakan tiga jenis komputer yang memiliki spesifikasi berbeda. Komputer yang digunakan memiliki spesifikasi yang dapat mewakili dari masing-masing periode komputer, hal ini untuk menguji apakah metode yang diusulkan dapat berjalan dengan baik pada komputer spesifikasi tinggi,rendah dan sedang. Kemudian dari masing-masing spesifikasi komputer dilakukan 10 kali percobaan sehingga menghasilkan total 30 kali percobaan. Adapun ketiga spesifikasi komputer yang digunakan tertera pada Tabel 2.

Tabel 2 Spesifikasi Komputer Untuk Uji Coba Metode Terhadap Data Uji

Spesifikasi	Prosesor	Memori	Sistem Operasi
Tinggi	Core I3-2330M 2.20 GHz	2.00 GB	Windows 7
Sedang	Core 2 duo 2.10 GHz	1.99 GB	Windows XP
Rendah	Pentium 4 2.4 GHz	1 GB	Windows XP

Pengujian dengan tiga spesifikasi komputer diatas dilakukan terhadap masing-masing *method* yang dimiliki metode JCCL dan JCCL+JDC yaitu (doUpdate,reload dan loadClass). Adapun hasil pengukuran efisiensi dari masing masing *method* adalah sebagai berikut:

4.1 doUpdate

Berdasarkan pengujian yang telah dilakukan terhadap *method* doUpdate yang dilakukan menggunakan komputer spesifikasi tinggi, sedang dan rendah didapatkan hasil bahwa grafik *method* doUpdate dari metode JCCL+JDC lebih stabil dibanding dengan grafik hasil *method* doUpdate dari method JCCL yang terus naik. Hal tersebut disebabkan karena pada metode JCCL *method* doUpdate akan mengeksekusi seluruh *method* yang terdapat di dalam kelas, sehigga jika ditambah *method* maka waktu eksekusi akan terus meningkat. Hal demikian tidak terjadi pada metode yang diusulkan yaitu JCCL+JDC, pada metode JCCL+JDC *method* yang dieksekusi hanyalah *method* yang ditandai sebagai *method* baru saja (Kazi, Chen, Stanley, & Lilja, 2000). Sehingga waktu eksekusi yang dihasilkan tergantung dengan alghoritma masing-masing *method,* jika *method* yang akan dieksekusi memiliki alghoritma yang lebih panjang dari *method* sebelumnya maka waktu yang dihasilkan akan lebih tinggi.

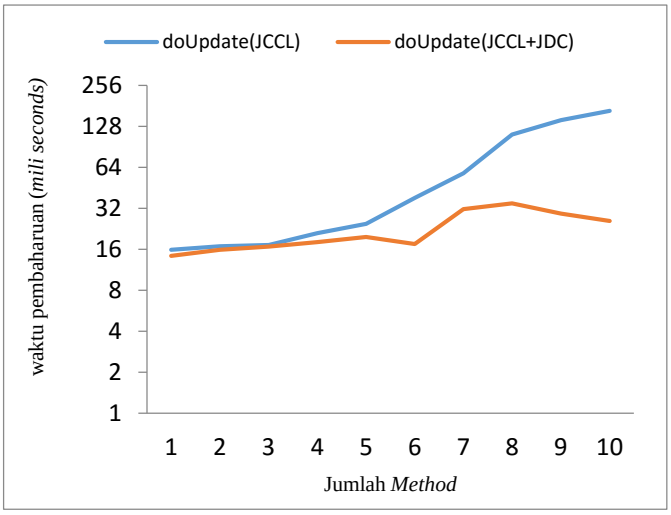
4.1.1 Komputer Spesifikasi Tinggi

Dalam pengujian yang dilakukan menggunakan komputer spesifikasi tiggi terhadap *method* doUpdate, didapatkan hasil bahwa waktu yang dihasilkan metode yang diusulkan yaitu JCCL+JDC memiliki waktu yang lebih efisien. Sedangkan metode JCCL sebelum dioptimalkan dengan metode JDC memiliki waktu yang terus meningkat. Hasil pengujian ini terlihat seperti ada Tabel 3 serta grafik perbedaan kedua *method* terlihat pada Gambar 5.

Tabel 3 Hasil Eksperimen *Method* doUpdate Pada Komputer Spesifikasi Tinggi

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	doUpdate	**doUpdate**
1	15,87	14,3
2	16,8	15,8
3	17,2	16,7
4	21	18
5	24,6	19,7
6	38	17,5
7	57,9	31,5
8	111	34,7
9	142	29,2
10	166	25,8

Method doUpdate dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih tinggi jika dilihat setiap kali pengujian dengan menambahkan jumlah method. Akan tetapi jika dilihat dari keseluruhan pengujian, *method* doUpdate memiliki fluktuasi waktu yang lebih stabil sedangkan method doUpdate dari metode JCCL memiliki fluktuasi waktu yang terus meningkat. Pada Tabel 3 diatas, terlihat bahwa pada percobaan yang ke-enam dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah hal tersebut dikarenakan method yang ke-enam memiliki alur yang lebih singkat dibandingkan method yang lainnya. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 0,63% dimana *method* JCCL+JDC lebih unggul.



Gambar 5 Grafik Perbandingan *method* doUpdate Pada Komputer Spesifikasi Tinggi

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 4 Nilai t hitung yang dihasilkan adalah 2,353 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,043 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara method doUpdate pada masing-masing metode.

Tabel 4 Hasil Uji Beda *Method* doUpdate Pada Komputer Spesifikasi Tinggi

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	38.71700	52.03145	16.45379	1.49594	75.93806	2.353	9	.043

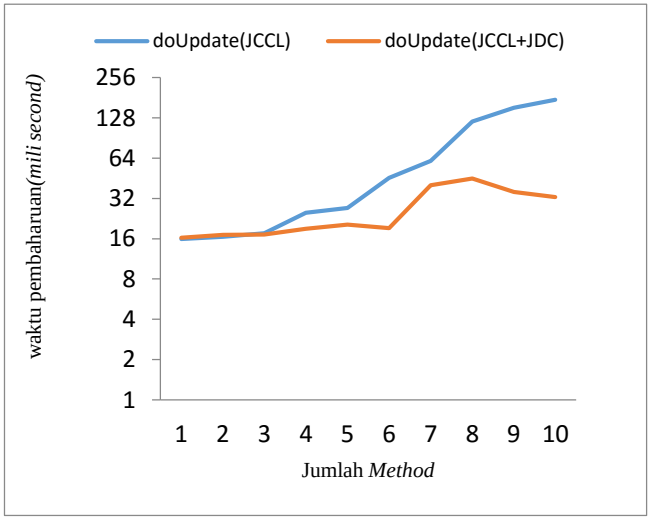
4.1.2 Komputer Spesifikasi Sedang

Metode yang diusulkan yaitu JCCL+JDC juga menunjukkan hasil yang bagus Gambar 6 ketika dilakukan pengujian menggunakan komputer spesifikasi sedang terhadap *method* doUpdate. Pada Gambar 6, grafik metode JCCL+JDC memiliki waktu yang lebih efisien dibanding metode JCCL pada setiap kali jumlah *method* ditambahakan. Hasil pengujian ini terlihat seperti ada Tabel 5 serta grafik perbedaan kedua *method* terlihat pada Gambar 6.

Tabel 5 Hasil Eksperimen *Method* doUpdate Pada Komputer Spesifikasi Sedang

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	doUpdate	**doUpdate**
1	15,87	16,3
2	16,5	17,1
3	17,6	17,2
4	25	19
5	27,2	20,4
6	45,7	19,2
7	61,2	40,2
8	120	45,1
9	152	35,7
10	175	32,8

Method doUpdate dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih tinggi jika dilihat setiap kali pengujian dengan menambahkan jumlah *method*. Akan tetapi jika dilihat dari keseluruhan pengujian, *method* doUpdate memiliki fluktuasi waktu yang lebih stabil sedangkan *method* doUpdate dari metode JCCL memiliki fluktuasi waktu yang terus meningkat. Pada Tabel 5 , terlihat bahwa pada percobaan yang ke enam dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah hal tersebut dikarenakan *method* yang ke enam memiliki alur yang lebih singkat dibandingkan *method* yang lainnya. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 0,59% dimana *method* JCCL+JDC lebih unggul seperti yang terlihat pada Gambar 6.



Gambar 6 Grafik Perbandingan *method* doUpdate Pada Komputer Spesifikasi Sedang

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan pada Tabel 6 Nilai t hitung yang dihasilkan adalah 2,352 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,043 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara *method* doUpdate pada masing-masing metode. Hasil pengujian ini terlihat seperti pada Tabel 6 serta grafik perbedaan kedua *method* terlihat pada Gambar 6.

Tabel 6 Hasil Uji Beda *Method* doUpdate Pada Komputer Spesifikasi Sedang

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	39.30700	52.84393	16.71072	1.50473	77.10927	2.352	9	.043

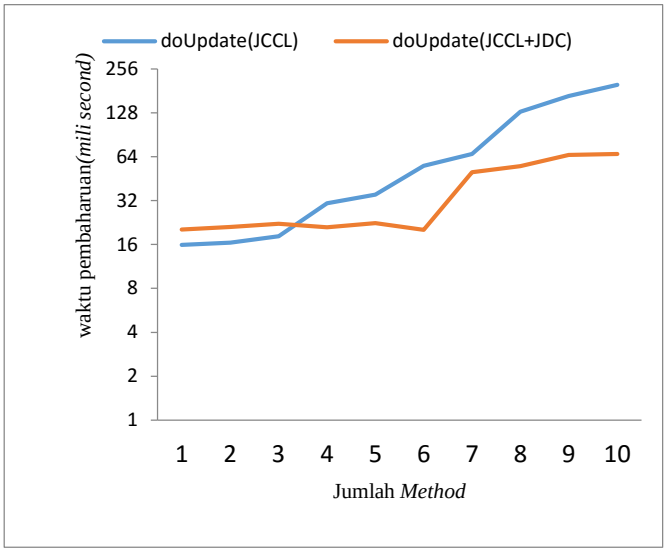
4.1.3 Komputer Spesifikasi Rendah

Pada pengujian yang dilakukan terhadap komputer spesifikasi rendah, metode yang diusulkan yaitu JCCL+JDC memiliki waktu yang kurang efisien pada awal pengujian tambar 7, hal tersebut disebabkan karena spesifikasi komputer yang rendah digunakan untuk memperbaharui *method* yang memiliki alghoritma panjang. Namun ketikan jumlah *method* terus ditambahkan, metode yang diusulkan yaitu JCCL+JDC menunjukkan hasil waktu yang bagus dibandingkan metode JCCL sebelum dioptimalkan dengan JDC, hal ini disebabkan karena metode yang diusulkan memiliki kemampuan untuk mengeksekusi hanya *method* baru saja.

Tabel 7 Hasil Eksperimen *Method* doUpdate Pada Komputer Spesifikasi Redah

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	doUpdate	**doUpdate**
1	15,87	20,3
2	16,5	21,1
3	18,2	22,2
4	30,7	21
5	35,2	22,4
6	55,5	20,2
7	66,9	50,2
8	130	55,1
9	167	65,7
10	199	66,8

Method doUpdate dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih tinggi jika dilihat setiap kali pengujian dengan menambahkan jumlah *method*. Akan tetapi jika dilihat dari keseluruhan pengujian, *method* doUpdate memiliki fluktuasi waktu yang lebih stabil sedangkan *method* doUpdate dari metode JCCL memiliki fluktuasi waktu yang terus meningkat. Pada Tabel 7 , terlihat bahwa pada percobaan yang ke 4,5 dan 6 dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah hal tersebut dikarenakan *method* yang ke-empat memiliki alur yang lebih singkat dibandingkan *method* yang lainnya. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 0,50% dimana *method* JCCL+JDC lebih unggul seperti yang terlihat pada Gambar 7.



Gambar 7 Grafik Perbandingan *method* doUpdate Pada Komputer Spesifikasi rendah

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 8 nilai t hitung yang dihasilkan adalah 2,393 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,040 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara method doUpdate pada masing-masing metode.

Tabel 8 Hasil Uji Beda *Method* doUpdate Pada Komputer Spesifikasi Rendah

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	36.98700	48.88519	15.45885	2.01664	71.95736	2.393	9	.040

4.2 Reload

Hasil pengujian yang dilakukan terhadap *method* reload menggunakan komputer spesifikasi tinggi, sedang dan rendah menunjukkan hasil bahwa grafik *method* reload pada metode JCCL memiliki waktu yang lebih efisien dibanding *method* reload pada metode JCCL+JDC. Hal tersebut disebabkan karena *method* reload pada metode JCCL+JDC memiliki alghoritma tambahan untuk mengecek versi dari *method* yang akan diperbaharui sehingga mengkonsumsi waktu untuk melakukan hal tersebut. Sedangkan *method* reload pada metode JCCL tidak memiliki alghoritma tambahan untuk mengecek versi dari *method* yang akan diperbaharui, sehingga waktu yang dihasilkan lebih efisien.

4.2.1 Komputer Spesifikasi Tinggi

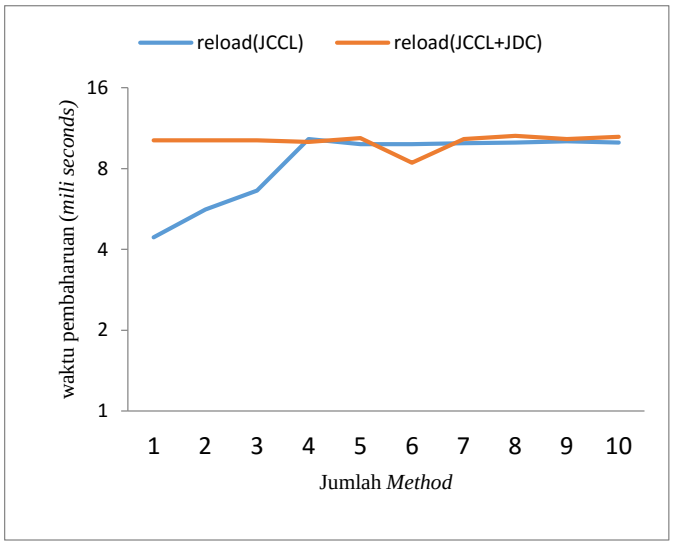
Pada pengujian menggunakan komputer spesifikasi tinggi, seperti terlihat pada Gambar 8. metode JCCL lebih baik dalam konsumsi waktu dibandingkan dengan metode yang diusulkan yaitu JCCL+JDC. Hal tersebut dikarenakan pada metode JCCL+JDC terdapat alghoritma untuk mengecek versi dari *method* yang akan diperbaharui. Sehingga alghoritma ini akan

mengkonsumsi waktu lebih lama dibanding metode JCCL sebelum ditambahkan metode JDC. Hasil pengujian ini terlihat seperti ada tabel 9 serta grafik perbedaan kedua *method* terlihat pada gambar 8.

Tabel 9 Hasil Eksperimen *Method* reload Pada Komputer Spesifikasi Tinggi

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	reload	**reload**
1	4,44	10,2
2	5,64	10,2
3	6,63	10,2
4	10,3	10,04
5	9,86	10,4
6	9,87	8,41
7	9,95	10,3
8	9,99	10,6
9	10,1	10,3
10	10	10,5

Method reload dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebi tinggi, hal ini dikarenakan didalam *method* reload pada penelitian ini ditambahkan dengan metode JDC agar proses pembaharuan yang dilakukan *method* doUpdate lebih efisien. Pada Tabel 9, terlihat bahwa pada percobaan yang ke enam dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah hal terseut dikarenakan *method* yang ke empat memiliki alur yang lebih singkat dibandingkan *method* yang lainnya. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 72 % dimana *method* JCCL lebih unggul karena tidak memiliki alghoritma untuk mengecek versi *method*.



Gambar 8 Grafik Perbandingan *method* reload Pada Komputer Spesifikasi Tinggi

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 10 nilai t hitung yang dihasilkan adalah 1,861 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,096 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara *method* reload pada masing-masing metode.

Tabel 10 Hasil Uji Beda *Method* reload Pada Komputer Spesifikasi Tinggi

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	-1.39200	2.36567	.74809	-3.08430	.30030	-1.861	9	.096

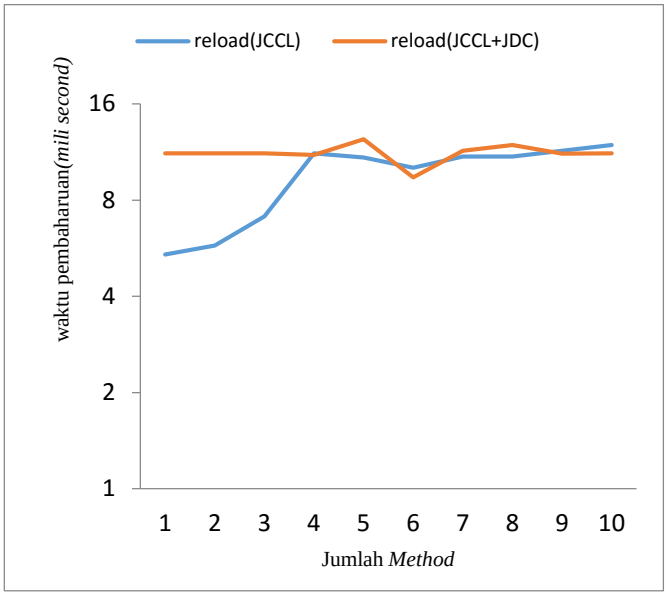
4.2.2 Komputer Spesifikasi Sedang

Hal yang sama terjadi ketika pengujian dilakukan pada komputer spesifikasi sedang. Pada pengujian menggunakan komputer spesifikasi sedang, seperti terlihat pada Gambar 9. metode JCCL lebih baik dalam konsumsi waktu dibandingkan dengan metode yang diusulkan yaitu JCCL+JDC. Perbedaan hanya terjadi pada waktu yang dihasilkan. Waktu yang dihasilkan lebih tinggi dibanding hasil waktu yang dihasilkan dari komputer spesifikasi tinggi, hal ini disebabkan karena spesifikasi komputer yang digunakan lebih rendah. Hasil pengujian ini terlihat seperti pada Tabel 11 serta grafik perbedaan kedua *method* terlihat pada Gambar 9.

Tabel 11 Hasil Eksperimen *Method* reload Pada Komputer Spesifikasi Sedang

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	reload	**reload**
1	5,41	11,2
2	5,77	11,2
3	7,12	11,2
4	11,2	11,08
5	10,86	12,4
6	10,1	9,42
7	10,94	11,4
8	10,95	11,9
9	11,4	11,17
10	11,9	11,2

Method reload dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih tinggi, hal ini dikarenakan didalam *method* reload pada penelitian ini ditambahkan dengan metode JDC agar proses pembaharuan yang dilakukan *method* doUpdate lebih efisien. Pada Tabel 7, terlihat bahwa pada percobaan yang ke enam dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah hal terseut dikarenakan *method* yang ke empat memiliki alur yang lebih singkat dibandingkan *method* yang lainnya. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 79,13 % dimana *method* JCCL lebih unggul karena tidak memiliki alghoritma untuk mengecek versi *method*.



Gambar 9 Grafik Perbandingan *method* reload Pada Komputer Spesifikasi Sedang

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan tabel 12 nilai t hitung yang dihasilkan adalah 2,078 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,067 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara method reload pada masing-masing metode.

Tabel 12 Hasil Uji Beda *Method* reload Pada Komputer Spesifikasi Sedang

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference				
					Lower	Upper			
Pair 1	jccl - jccljdc	-1.65200	2.51373	.79491	-3.45022	.14622	-2.078	9	.067

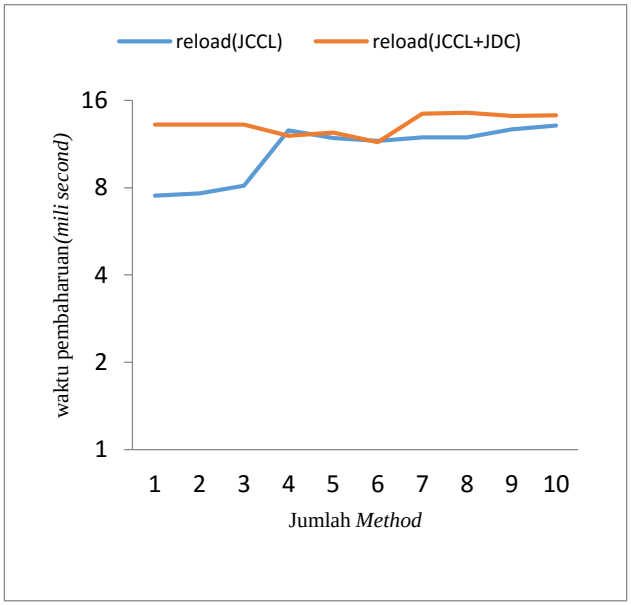
4.2.3 Komputer Spesifikasi Rendah

Hasil pengujian yang dilakukan dengan menggunakan komputer spesifikasi rendah Gambar 10, menghasilkan waktu yang lebih tinggi dibandingkan hasil pengujian menggunakan komputer spesifikasi tinggi dan sedang. Hal ini karena spesifikasi komputer yang digunakan lebih rendah dibanding spesifikasi komputer yang digunakan untuk menguji sebelumnya yaitu tinggi dan rendah. Hasil pengujian ini terlihat seperti pada Tabel 13 serta grafik perbedaan kedua *method* terlihat pada Gambar 10.

Tabel 13 Hasil Eksperimen *Method* reload Pada Komputer Spesifikasi Rendah

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	reload	**reload**
1	7,51	13,2
2	7,65	13,2
3	8,12	13,2
4	12,6	12,09
5	11,86	12,4
6	11,6	11,47
7	11,94	14,4
8	11,92	14,5
9	12,7	14,15
10	13,1	14,2

Method reload dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih tinggi, hal ini dikarenakan didalam *method* reload pada penelitian ini ditambahkan dengan metode JDC agar proses pembaharuan yang dilakukan *method* doUpdate lebih efisien. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 85,19 % dimana *method* JCCL lebih unggul karena tidak memiliki alghoritma untuk mengecek versi *method*.



Gambar 10 Grafik Perbandingan *method* reload Pada Komputer Spesifikasi rendah

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 14 nilai t hitung yang dihasilkan adalah 3,231 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,010 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara *method* reload pada masing-masing metode.

Tabel 14 Hasil Uji Beda *Method* reload Pada Komputer Spesifikasi Rendah

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	-2.38100	2.33053	.73698	-4.04816	-.71384	-3.231	9	.010

4.3 loadClass

Method loadClass ini bertugas untuk memasukkan kelas yang sudah diperbaharui kedalam JVM. Berdasarkan hasil percobaan yang dilakukan menggunakan komputer spesifikasi Tinggi,sedang dan rendah, didapatkan hasil bahwa grafik *method* loadClass pada metode yang diusulkan yaitu JCCL+JDC memiliki waktu yang lebih rendah atau efisien pada pengujian menggunakan komputer spesifikasi tinggi,sedang dan rendah. Kelas ini memiliki tugas yang ringan karena hanya bertugas memasukkan kelas yang berisi *method* yang telah diperbaharui ke dalam JVM. *Method* loadClass pada metode yang diusulkan yaitu JCCL+JDC memiliki waktu yang lebih unggul karena pada metode yang diusulkan, kelas yang dimasukkan hanya berisi *method* baru saja.

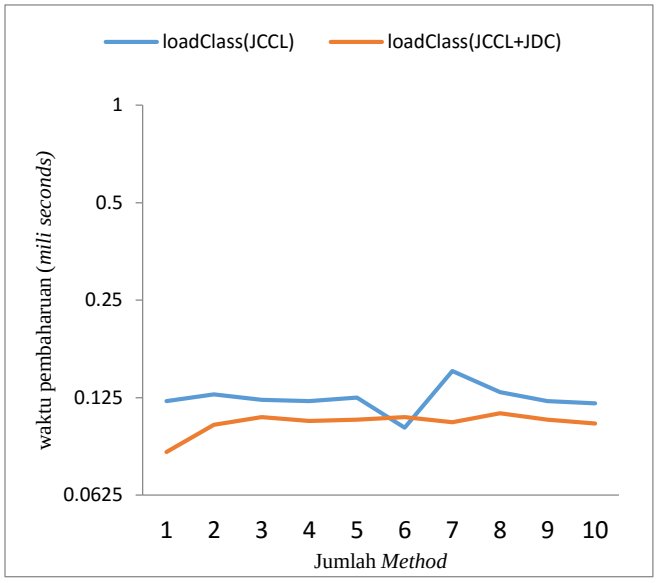
4.3.1 Komputer Spesifikasi Tinggi

Pengujian yang dilakukan menggunakan komputer spesifikasi tinggi Gambar 11, menunjukkan hasil bahwa metode JCCL+JDC memiliki fluktuasi waktu yang lebih baik dibanding metode JCCL. Hasil pengujian ini terlihat seperti pada Tabel 15 serta grafik perbedaan kedua *method* terlihat pada Gambar 11.

Tabel 15 Hasil Eksperimen *Method* loadClass Pada Komputer Spesifikasi Tinggi

Jumlah *Method*	**JCCL**	**JCCL+JDC**
	loadClass	**loadClas**
1	0,122	0,085
2	0,128	0,103
3	0,123	0,109
4	0,122	0,106
5	0,125	0,107
6	0,101	0,109
7	0,151	0,105
8	0,13	0,112
9	0,122	0,107
10	0,12	0,104

Method loadClass dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih tinggi tetapi ada juga yang memiliki waktu lebih rendah, hal ini dikarenakan *method* loadClass bertugas memuat ulang kelas yang diperbaharui ke dalam JVM dengan menyertakan metode tambahan (JDC) agar kelas yang dimuat hanyalah kelas yang baru saja. Waktu yang dihasilkan oleh loadClass tergantung dari alghoritma setiap *method* yang ditambahkan serta spesifikasi komputer yang digunakan untk percobaan. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 0,15% dimana *method* JCCL+JDC lebih unggul.



Gambar 11 Grafik Perbandingan *method* loadClass Pada Komputer Spesifikasi Tinggi

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 16 nilai t hitung yang dihasilkan adalah 4,371 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,002 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara method reload pada masing-masing metode.

Tabel 16 Hasil Uji Beda *Method* loadClass Pada Komputer Spesifikasi Tinggi

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	.01970	.01443	.00456	.00938	.03002	4.317	9	.002

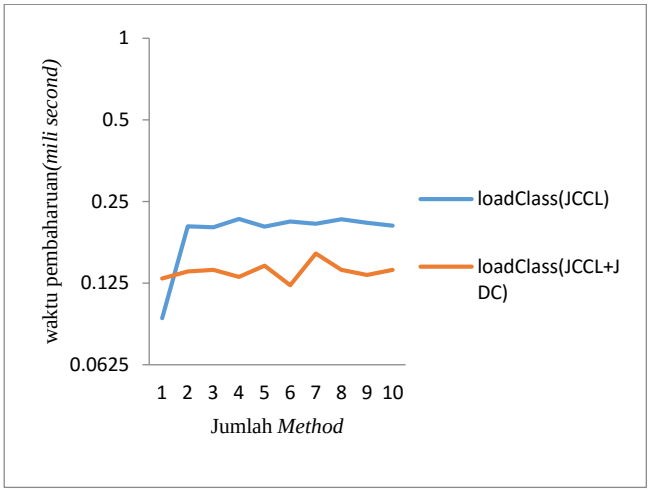
4.3.2 Komputer Spesifikasi Sedang

Pada pengujian *method* loadClass menggunakan komputer spesifikasi sedang Gambar 12, dihasilkan waktu dari metode yang diusulkan yaitu JCCL+JDC lebih unggul dibandingkan waktu metode JCCL. Hasil pengujian ini terlihat seperti pada Tabel 17 serta grafik perbedaan kedua *method* terlihat pada Gambar 12.

Tabel 17 Hasil Eksperimen *Method* loadClass Pada Komputer Spesifikasi Sedang

Jumlah *Method*	JCCL	JCCL+JDC
	loadClass	loadClass
1	0,093	0,13
2	0,203	0,138
3	0,201	0,14
4	0,216	0,132
5	0,202	0,145
6	0,211	0,123
7	0,207	0,161
8	0,215	0,14
9	0,209	0,134
10	0,204	0,14

Method loadClass dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah, hal ini dikarenakan *method* loadClass bertugas memuat ulang kelas yang diperbaharui ke dalam JVM dengan menyertakan metode tambahan (JDC) agar kelas yang dimuat hanyalah kelas yang baru saja. Waktu yang dihasilkan oleh loadClass tergantung dari alghoritma setiap *method* yang ditambahkan dan juga spesifikasi komputer yang digunakan untuk melakukan percobaan. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 0,29% dimana *method* JCCL+JDC lebih unggul.



Gambar 12 Grafik Perbandingan *method* loadClass Pada Komputer Spesifikasi Sedang

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 18 nilai t hitung yang dihasilkan adalah 5,132 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,001 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara method reload pada masing-masing metode.

Tabel 18 Hasil Uji Beda *Method* loadClass Pada Komputer Spesifikasi Sedang

	Paired Differences					t	df	Sig. (2-tailed)
	Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1 jccl - jccljdc	.05780	.03562	.01126	.03232	.08328	5.132	9	.001

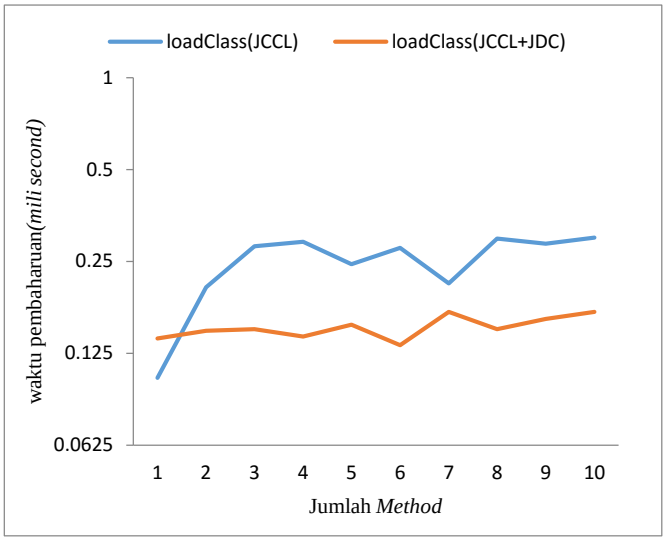
4.3.3 Komputer Spesifikasi Rendah

Pada pengujian *method* loadClass menggunakan komputer spesifikasi sedang Gambar 12, dihasilkan waktu dari metode yang diusulkan yaitu JCCL+JDC lebih unggul dibandingkan waktu metode JCCL. Pada pengujian yang dilakukan menggunakan komputer spesifikasi rendah ini grafik yang dihasilkan memiliki fluktuasi waktu yang kurang stabil dibandingkan pengujian menggunakan komputer spesifikasi rendah dan tinggi. Hasil pengujian ini terlihat seperti ada Tabel 19 serta grafik perbedaan kedua *method* terlihat pada Gambar 13.

Tabel 19 Hasil Eksperimen *Method* loadClass Pada Komputer Spesifikasi Rendah

Jumlah *Method*	JCCL	JCCL+JDC
	loadClass	loadClass
1	0,104	0,14
2	0,206	0,148
3	0,28	0,15
4	0,29	0,142
5	0,245	0,155
6	0,277	0,133
7	0,212	0,171
8	0,297	0,15
9	0,286	0,162
10	0,299	0,171

Method loadClass dari metode yang diusulkan (JCCL+JDC) memiliki waktu yang lebih rendah, hal ini dikarenakan *method* loadClass bertugas memuat ulang kelas yang diperbaharui ke dalam JVM dengan menyertakan metode tambahan (JDC) agar kelas yang dimuat hanyalah kelas yang baru saja. Waktu yang dihasilkan oleh loadClass tergantung dari alghoritma setiap *method* yang ditambahkan dan juga spesifikasi komputer yang digunakan untuk melakukan percobaan. Selisih total waktu dari kesepuluh percobaan yang dilakukan adalah 0,39% dimana *method* JCCL+JDC lebih unggul.



Gambar 13 Grafik Perbandingan *method* loadClass Pada Komputer Spesifikasi rendah

Untuk memvalidasi perbedaan yang terjadi maka dilakukan uji beda dengan metode t tes. Berdasarkan Tabel 20 nilai t hitung yang dihasilkan adalah 5,127 pada derajat bebas 9 lebih besar daripada nilai t tabel sebesar 1,761. nilai sig.2-tailed lebih kecil daripada nilai kritik 0,05 (0,001 < 0,05) berarti H0 dapat ditolak, artinya terdapat perbedaan antara method reload pada masing-masing metode.

Tabel 20 Hasil Uji Beda *Method* loadClass Pada Komputer Spesifikasi Rendah

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	jccl - jccljdc	.09740	.06008	.01900	.05442	.14038	5.127	9	.001

5 KESIMPULAN

Dari hasil penelitian yang dilakukan mulai dari tahap awal hingga proses pengujian, dapat disimpulkan bahwa Jumlah *method* yang terus bertambah pada perangkat lunak yang akan diperbaharui mempengaruhi efisiensi proses pembaharuan pada metode JCCL. Hal ini terbukti berdasarkan hasil pengujian metode JCCL bahwa grafik yang dihasilkan metode JCCL terus meningkat seiring bertambahnya jumlah *method*.

Dengan menerapkan metode JDC pada metode JCCL maka efisiensi proses pembaharuan perangkat lunak pada saat *runtime* meningkat walaupun jumlah *method* terus bertambah, hal ini menjawab pertanyaan penelitian: Apakah dengan menerapkan metode JDC pada metode JCCL dapat meningkatkan efisiensi proses pembaharuan perangkat lunak pada saat runtime apabila jumlah *method* terus bertambah. Hal tersebut didukung dari hasil pengujian metode yang diusulkan yaitu JCCL+JDC, dimana setiap pertambahan *method* memiliki fluktuasi waktu yang lebih stabil dibandingkan pertambahan *method* yang dilakukan pada metode JCCL dimana pada metode JCCL waktu yang dibutuhkan untuk melakukan proses pembaharuan terus meningkat.

Berdasarkan uji t yang telah dilakukan terhadap metode yang diuji yaitu JCCL dan JCCL+JDC yang dilakukan menggunakan komputer spesifikasi tinggi,sedang dan rendah serta dilakukan percobaan sebanyak 30 kali yang dilakukan pada tiga unit komputer dengan spesifikasi yang berbeda, dimana pada setiap komputer dilakukan penambahan *method* setiap kali percobaan dilakukan, maka didapat data perbedaan yang siknifikan antara kedua metode tersebut. Dalam hal ini metode JCCL+JDC memiliki waktu yang lebih efisien walaupun jumlah *method* terus bertambah.

DAFTAR REFERENSI

Fong, Y. L. A. S. (2012). Heuristic optimisation algorithm for Java dynamic compilation. *Engineering and Technology*, (February), 307–312. doi:10.1049/iet-sen.2011.0144

Gregersen, A. R., & Koskimies, K. (2012). Javeleon : An Integrated Platform for Dynamic Software Updating and its Application in Self- * systems. *Transactions on Software Engineering*.

Hayden, C. M., Smith, E. K., & Foster, J. S. (2012). Kitsune : Efficient , General-purpose Dynamic Software Updating for C. *OOPSLA*.

Kazi, I. H., Chen, H. H., Stanley, B., & Lilja, D. J. (2000). Techniques for obtaining high performance in Java programs. *ACM Computing Surveys*, *32*(3), 213–240.

Kim, D. K., Tilevich, E., & J, C. (2010). Dynamic Software Updates for Parallel High Performance Applications. *Dept. of Computer Science*.

Orso, A., Rao, A., & Harrold, M. (2002). A Technique for Dynamic Updating of Java Software. *18th IEEE International Conference on Software Maintenance*, *pp649658*.

Pukall, M., K, C., G, S., Grebhahn, A., Schr, R., & Saake, G. (2011). J AV A DAPTOR – Flexible Runtime Updates of Java Applications. *Softw. Pract. Exper.*, 1–33. doi:10.1002/spe

Seifzadeh, H., Abolhassani, H., & Moshkenani, M. S. (2012). A survey of dynamic software updating. *Wiley Online Lirary*. doi:10.1002/smr

Würthinger, T., Wimmer, C., & Stadler, L. (2013). Unrestricted and safe dynamic code evolution for Java ☆. *Science of Computer Programming*, *78*(5), 481–498. doi:10.1016/j.scico.2011.06.005

Zhang, S. (2007). Type-Safe Dynamic Update Transaction. *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*, (60673116).

BIOGRAFI PENULIS

Tory Ariyanto. Menyelesaikan pendidikan S1 Teknik Informatika di Sekolah Tinggi Managemen Informatika (STMIK) Widya Pratama Pekalongan Indonesia, S2 Magister Teknik Informatika di Universitas Dian Nuswantoro Semarang Indonesia. Saat ini menjadi Dosen Pemrograman Komputer di STMIK Widya Pratama Pekalongan Indonesia. Minat penelitian saat ini adalah Software enginering.

Romi Satria Wahono. Menempuh pendidikan S1, S2 di bidang computer science di Saitama University, Jepang, dan Ph.D di bidang Software Engineering di Universiti Teknikal Malaysia Melaka. Saat ini menjadi dosen di Fakultas Ilmu Komputer di Universitas Dian Nuswantoro, Indonesia. Founder dan CEO PT Brainmatics Cipta Informatika, sebuah perusahaan pengembangan software di Indonesia. Minat penelitian saat ini adalah software engineering dan machine learning. Professional member dari ACM, PMI and IEEE.

Purwanto. Menempuh pendidikan S1 di universitas diponegoro Semarang Indonesia, S2 di **Purwanto**. Menempuh pendidikan S1 di universitas diponegoro Semarang Indonesia, S2 di Sekolah Tinggi Teknologi Informasi Benarif Indonesia dan Ph.D di multimedia university Malaysia. Saat ini menjadi dosen pascasarjana Magister Teknik Informatika di Universitas Dian Nuswantoro Semarang, Indonesia.