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.

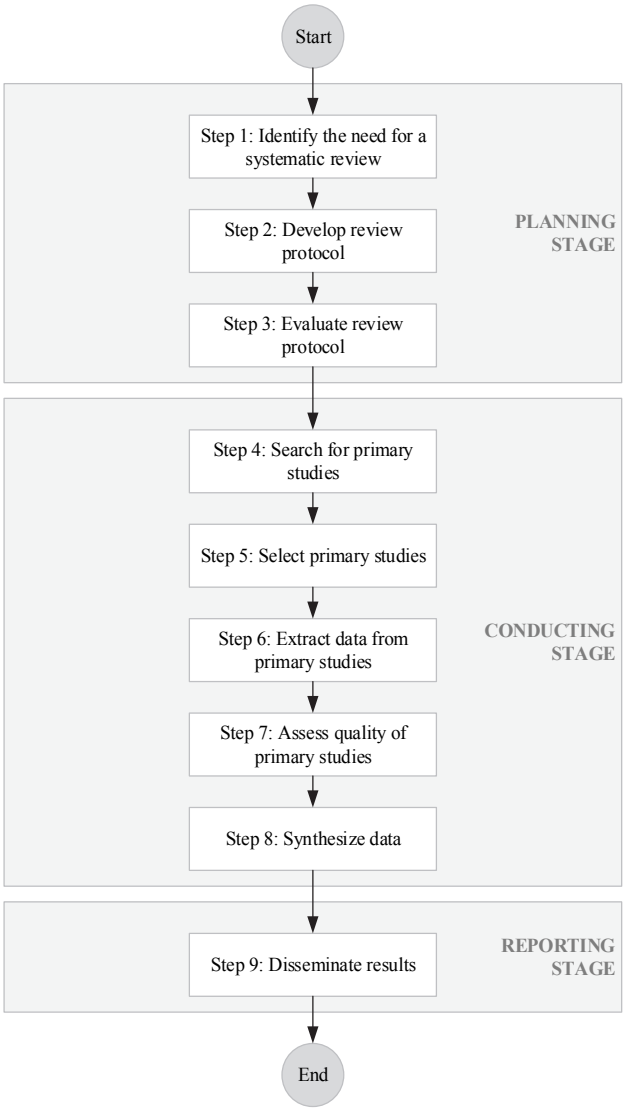

Figure 1 Systematic Literature Review Steps

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.

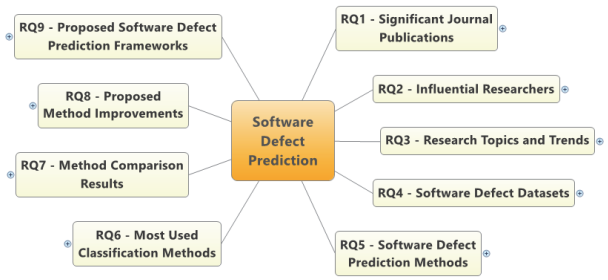

Figure 2 Basic Mind Map of the SLR on Software Defect Prediction

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.

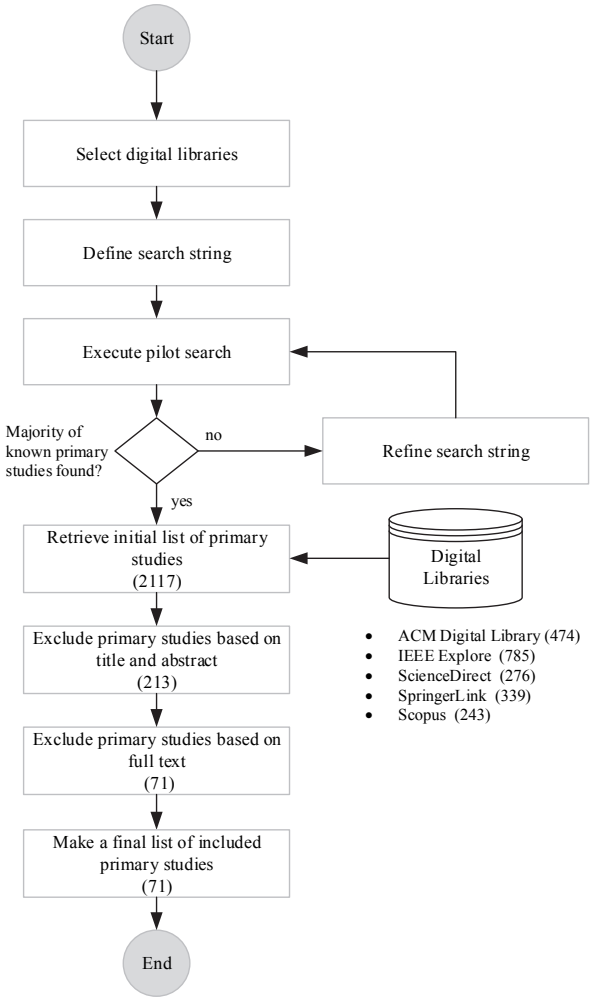

Figure 3 Search and Selection of Primary Studies

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.

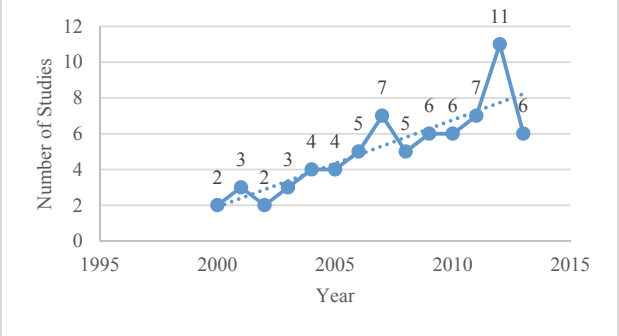

Figure 4 Distribution of Selected Studies over the Years

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.

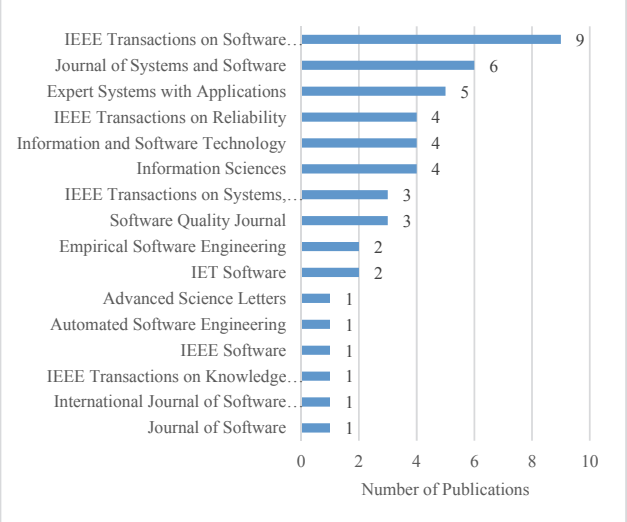

Figure 5 Journal Publications and Distribution of Selected Studies

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.

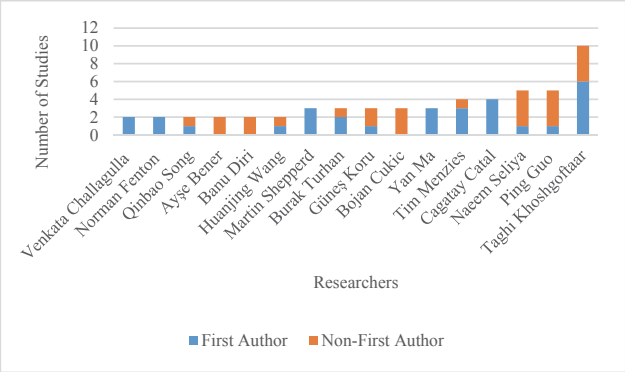

Figure 6 Influential Researchers and Number of Studies

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.

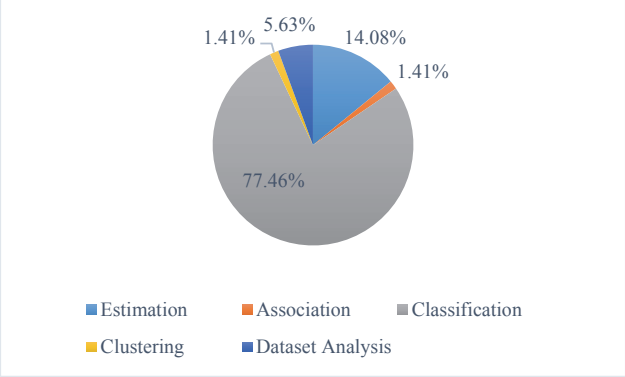

Figure 7 Distribution of Research Topics

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.

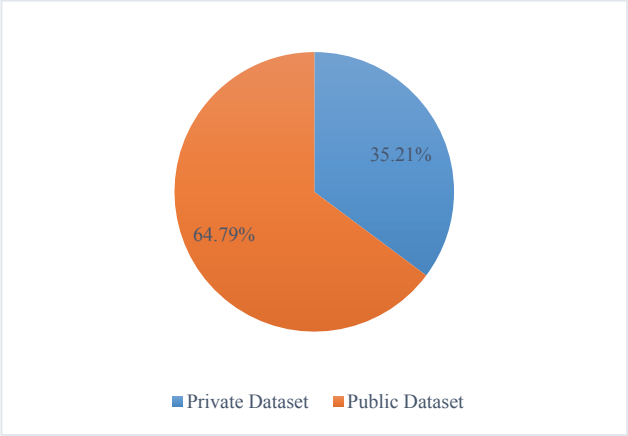

Figure 8 Total Distribution of Datasets

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.

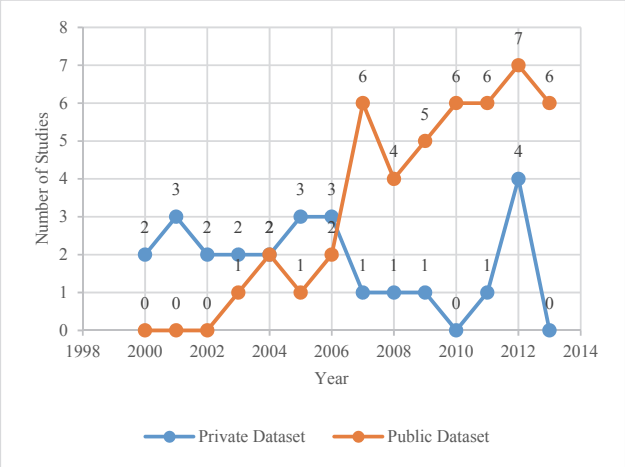

Figure 9 Distribution of Private and Public Datasets

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.

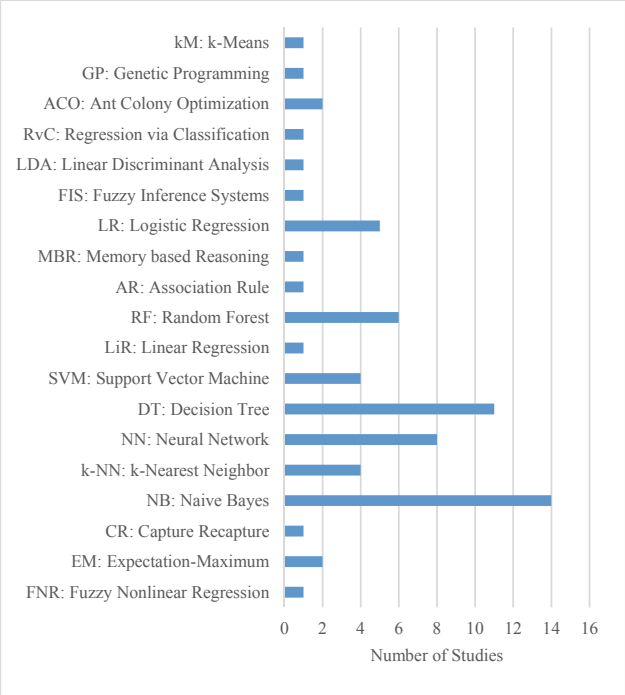

Figure 10 Methods Used in Software Defect Prediction

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)

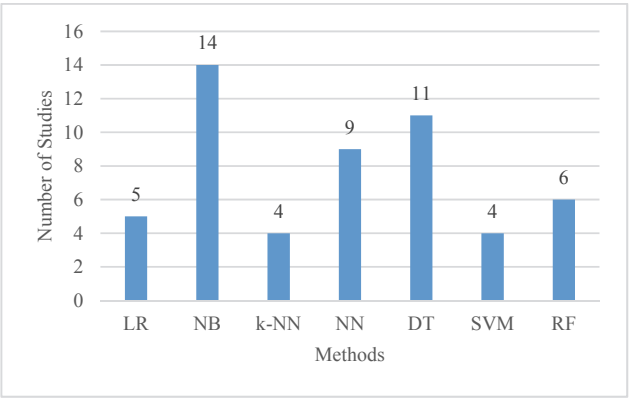

Figure 11 Most Used Methods in Software Defect Prediction

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.

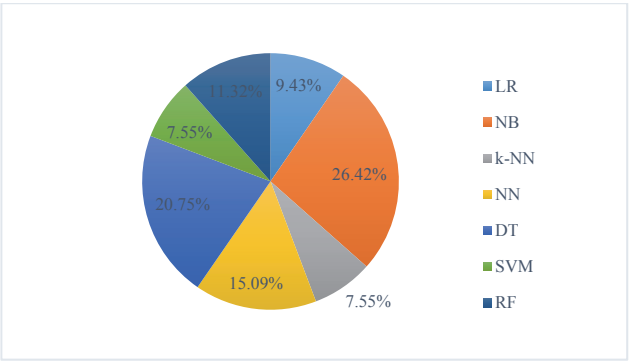

Figure 12 Distribution of the Studies over Type of Methods

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.

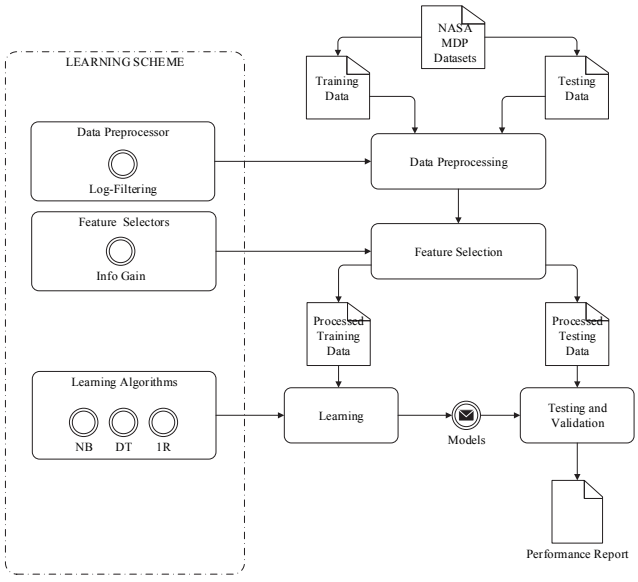

Figure 13 Menzies *et al.*'s Framework
(Compiled from (Menzies et al., 2007))

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.

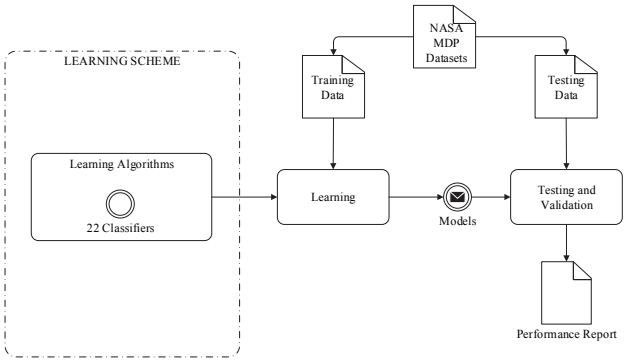

Figure 14 Lessmann *et al.*'s Framework
(Compiled from (Lessmann et al., 2008))

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.

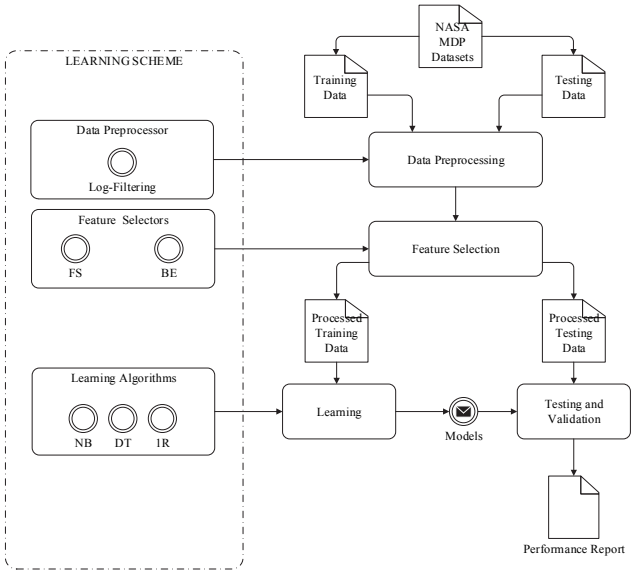

Figure 15 Song *et al.*'s Framework
(Compiled from (Song et al., 2011))

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

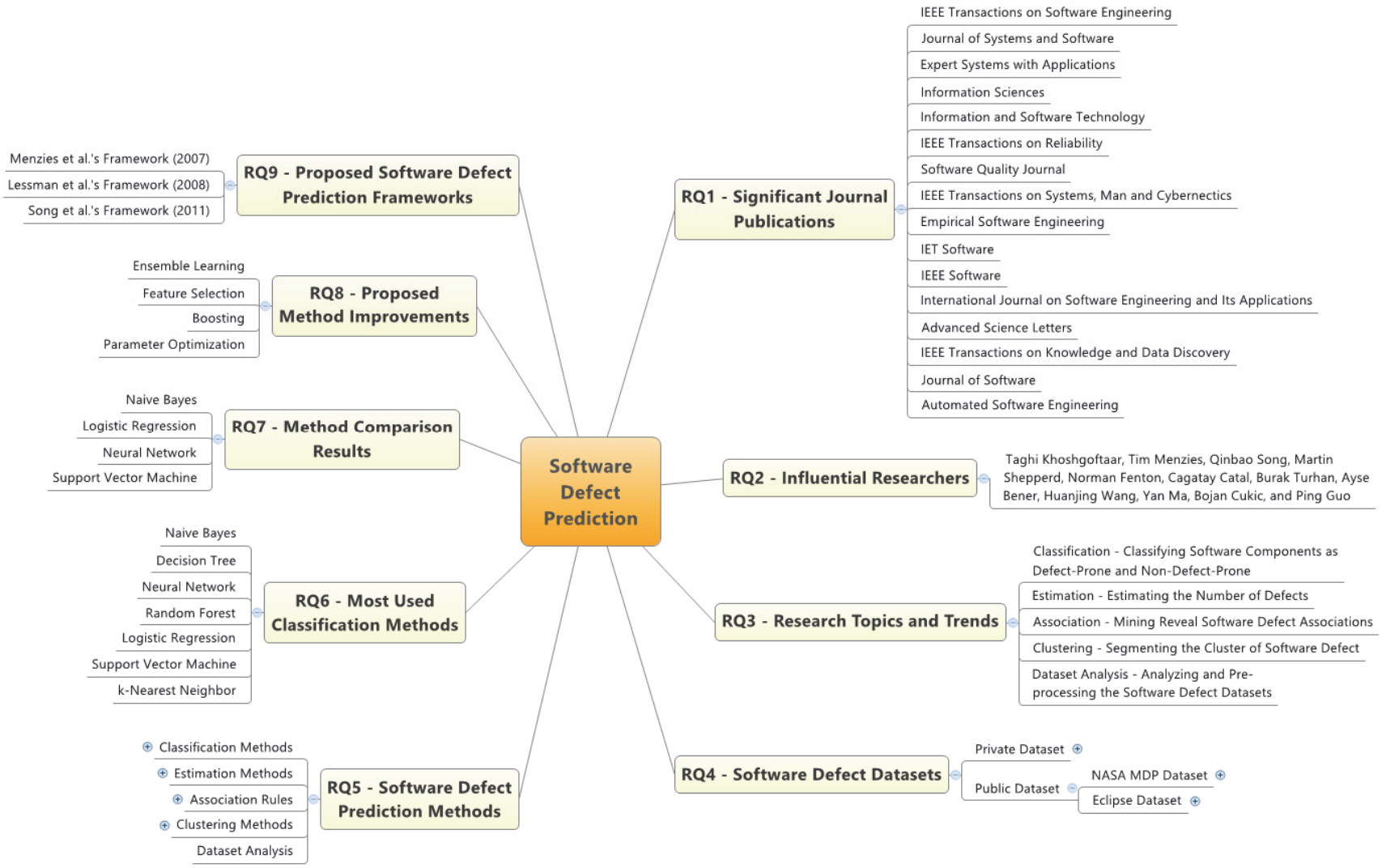

Figure 16 Complete Mind Map of the SLR on Software Defect Prediction

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.