A Systematic Review of Fault Prediction Performance in Software Engineering

Tracy Hall
Brunel University

Sarah Beecham
Lero – The Irish Software Engineering Research Centre
University of Limerick, Ireland

David Bowes
University of Hertfordshire

Steve Counsell
Brunel University

August 2011

Contact

Address	Lero Tierney Centre Building University of Limerick Ireland
Phone	+353 61 233799
Fax	+353 61 213036
E-Mail	info@lero.ie
Website	http://www.lero.ie/

This work is partially supported by Science Foundation Ireland under grant no. 03/CE2/I303-1

Lero Technical Report Lero-TR-2011-03

A Systematic Review of Fault Prediction Performance in Software Engineering

Tracy Hall[a], Sarah Beecham[b], David Bowes[c], David Gray[c], Steve Counsell[a]

[a]Brunel University, Uxbridge, Middlesex, UB8 3PH UK

[b]Lero, The Irish Software Engineering Research Centre,Univ of Limerick, Limerick, Ireland

[c]University of Hertfordshire, Hatfield, Hertfordshire, AL10 9AB UK

Email: [tracy.hall; steve.counsell]@brunel.ac.uk; sarah.beecham@lero.ie; [d.h.bowes; d.gray]@herts.ac.uk

Abstract

BACKGROUND – The accurate prediction of where faults are likely to occur in code is important since it can help direct test effort, reduce costs and improve the quality of software. As a consequence, many different fault prediction models have been developed and reported in the literature. However, there is no consensus on what constitutes effective fault prediction.
OBJECTIVE – To investigate how the context of models, the independent variables used and the modelling techniques applied influence the performance of fault prediction models.
METHOD – A systematic literature review identifying 208 fault prediction studies published from January 2000 to December 2010. A synthesis of the quantitative and qualitative results of those 35 studies which report sufficient contextual and methodological information according the criteria we develop and apply. This synthesis includes a detailed analysis of the relative predictive performances of 203 models (or model variants) reported in 18 of these 35 studies which allow us to calculate precision, recall and f-measure.
RESULTS –There are large variations in the performance of these 203 models. The models that perform well tend to be based on simple modelling techniques such as Naïve Bayes or Logistic Regression. Combinations of independent variables have been used by models that perform well. These combinations include product, process and people metrics. Feature selection has been applied to these combinations by models which are performing particularly well. In addition, such models tend to have been trained on large data sets which are rich with faults.
CONCLUSION – The methodology used to build models seems to be influential to predictive performance. Although there are a set of fault prediction studies in which confidence is possible, many open questions remain about effective fault prediction. More studies are needed that use a reliable methodology and which report their context, methodology and performance comprehensively. This would enable a meta-analysis across more studies. It would also produce models more likely to be used by industry.

1. Introduction

This Systematic Literature Review (SLR) aims to identify and analyse the models used to predict faults in source code in 208 studies published between January 2000 and December 2010. Our analysis investigates how model performance is affected by the context in which the model was developed, the independent variables used in the model and the technique on which the model was built. Our results enable researchers to develop prediction models based on best knowledge and practice across many previous studies. Our results also help practitioners to make effective decisions on prediction models most suited to their context.

Fault[1] prediction modelling has become a popular method for the early identification of fault-prone code. It is an important area of research and the subject of many previous studies. These studies typically produce fault prediction models which allow software engineers to focus development activities on fault-prone code, thereby improving software quality and making better use of resources. The many fault prediction models published are complex and disparate and no up-to-date comprehensive picture of the current state of fault prediction exists. Two previous reviews of the area have been performed ([1] and [2])[2]. Our review differs from these reviews in the following ways:

[1] The term 'fault' is used interchangeably in this study with the terms 'defect' or 'bug' to mean a static fault in software code. It does not denote a 'failure' (i.e. the possible result of a fault occurrence).

[2] Note that two referencing styles are used throughout this paper; [ref#] refers to papers in the main reference list. while [[ref#]] refers to papers in the separate systematic literature review list located after the main reference list.

- *Timeframes.* Our review is the most contemporary because it includes studies published from 2000-2010. Fenton and Neil conducted a critical review of software fault prediction research up to 1999 [1]. Catal and Diri's [2] review covers work published between 1990 and 2007.
- *Systematic approach.* We follow Kitchenham's [3] original and rigorous procedures for conducting systematic reviews. Catal and Diri did not report on how they sourced their studies stating that they adapted Jørgensen and Shepperd's [4] methodology. Fenton and Neil did not apply the systematic approach introduced by Kitchenham [3] as their study was published well before these guidelines were produced.
- *Comprehensiveness.* We do not rely on search engines alone and, unlike Catal and Diri, we read through relevant Journals and Conferences paper-by-paper. As a result, we analysed many more papers.
- *Analysis.* We provide a more detailed analysis of each paper. Catal and Diri focused on the context of studies including: where papers are published, year of publication, types of metrics used, datasets and approach. In addition, we report on the performance of models and synthesise the findings of studies.

We make four significant contributions by presenting:

1) A set of 208 studies addressing fault prediction in software engineering from January 2000 to December 2010. Other researchers can use these studies as the basis of future investigations into fault prediction.
2) A subset of 35 fault prediction studies which report sufficient contextual and methodological detail to enable these studies to be reliably analysed by other researchers and evaluated by model users planning to select an appropriate model for their context.
3) A set of criteria to assess that sufficient contextual and methodological detail is reported in fault prediction studies. We have used these criteria to identify the 35 studies mentioned above. They can also be used to guide other researchers to build credible new models that are understandable, usable, replicable and in which researchers and users can have a basic level of confidence. These criteria could also be used to guide Journal and Conference reviewers in determining that a fault prediction paper has adequately reported a study.
4) A synthesis of the current state-of-the-art in software fault prediction as reported in the 35 studies satisfying our assessment criteria. This synthesis is based on extracting and combining: qualitative information on the main findings reported by studies; quantitative data on the performance of these studies; detailed quantitative analysis of the 203 models (or model variants) reported in 18 studies which either report (or we can calculate from what is reported) precision, recall and f-measure performance data.

This paper is organised as follows. In the next section, we present our systematic literature review methodology. In Section Three, we present our criteria developed to assess whether or not a study reports sufficient contextual and methodological detail to enable us to synthesise a particular study. Section Four identifies the threats to validity of this study. Section Five shows the results of applying our assessment criteria to 208 studies. Section Six reports the results of extracting data from the 35 studies which satisfy our assessment criteria. Section Seven synthesises our results and Section Eight discusses the methodological issues associated with fault prediction studies. Finally, in Section Nine we summarise and present our conclusions.

2. Methodology

We take a systematic approach to reviewing the literature on the prediction of faults in code. Systematic literature reviews are well established in medical research and increasingly in software engineering. We follow the systematic literature review approach identified by Kitchenham and Charters [3].

2.1 Research Questions

The overall aim of this systematic literature review (SLR) is to analyse the models used to predict faults in source code. Our analysis is driven by the research questions in Table 1.

Research Questions		Motivation
RQ1	How does context affect fault prediction?	Context has been shown to be a key factor in the comparative use of software metrics in general [5]. Context is important in fault prediction modelling as it can affect the performance of models in a particular context and the transferability of models between contexts. Currently the impact context variables have on the transferability of models is not clear. This makes it difficult for potential model users to select models that will perform well in a particular context. We aim to present a synthesis of current knowledge on the impact of context on models and the transferability of models.
RQ2	Which independent variables should be included in fault prediction models?	There are a range of independent variables that have been used in fault prediction models. Currently the impact individual independent variables have on model performance is not clear. Although the performance of independent variables has been investigated within individual studies, no comparison of performance across studies has been done. This makes it difficult for model builders to make informed decisions about the independent variables on which to base their models. We aim to present a synthesis of current knowledge on the impact independent variables have on models.
RQ3	Which modeling techniques perform best when used in fault prediction?	Fault prediction models are based on a wide variety of both machine learning and regression modelling techniques. Currently the impact modelling technique has on model performance is not clear. Again, the performance of modelling techniques has been investigated within individual studies, but no comparison of performance across studies has been done. This makes it difficult for model builders to make effective technique selections. We aim to present a synthesis of current knowledge on the impact of modelling technique on model performance.

Table 1. The research questions addressed

2.2 Inclusion criteria

To be included in this review, a study must be reported in a paper published in English as either a Journal paper or Conference proceedings. The criteria for studies to be included in our SLR are based on the inclusion and exclusion criteria presented in Table 2.

Inclusion criteria	Exclusion criteria
A paper must be…	A paper must not be…
- An empirical study - Focused on predicting faults in units of a software system - Faults in code is the main output (dependent variable)	- Focused on: testing, fault injection, inspections, reliability modelling, aspects, effort estimation, debugging, faults relating to memory leakage, nano-computing, fault tolerance. - About the detection or localisation of existing individual known faults.

Table 2. Inclusion and exclusion criteria

Before accepting a paper into the review, we excluded repeated studies. If the same study appeared in several publications we included only the most comprehensive or most recent.

2.3 Identification of papers

Included papers were published between January 2000 and December 2010. Our searches for papers were completed at the end of May 2011and it is therefore unlikely that we missed any papers published in our time period as a result of publication time lags. There were four elements to our searches:

1. Key word searching using the search engines: ACM Digital Library, IEEExplore and the ISI Web of Science. These search engines covered the vast majority of software engineering publications and the search string we used is given in Appendix A.
2. An issue-by-issue manual reading of paper titles in relevant Journals and Conferences. The Journals and Conferences searched are shown in Appendix B. These were chosen as highly relevant software engineering publications found previously to be good sources of software engineering research [4].
3. A manual search for publications from key authors using DBLP. These authors were selected as appearing most frequently in our initial list of papers and are: Khoshgoftaar, Menzies, Nagappan, Ostrand and Weyuker.
4. The identification of papers using references from included studies.

Table 3 shows that our initial searches elicited 2,073 papers. The title and abstract of each was evaluated and 1,762 were rejected as not relevant to fault prediction. This process was validated using a randomly selected 80 papers from the initial set of 2,073. Three researchers separately interpreted and applied the inclusion and exclusion criteria to the 80 papers. Pairwise inter-rater reliability was measured across the three sets of decisions to get a fair/good agreement on the first iteration of this process. On the basis of the disagreements we clarified our inclusion and exclusion criteria. A second iteration resulted in 100% agreement between the three researchers.

We read the remaining 311 papers in full. This resulted in a further 178 papers being rejected. An additional 80 secondary papers were identified from references and after being read in full, accepted into the included set. We also included two extra papers from Catal and Diri's [2] review which overlapped our timeframe. Our initial searches omitted these two of Catal and Diri's papers as their search terms included the word 'quality'. We did not include this word in our searches as it generates a very high false positive rate. This process resulted in the 208 papers included in this review.

Selection Process	**# of papers**	**Validation**
Papers extracted from databases, Conferences and Journals	2,073	80 random papers independently classified by 3 researchers
Sift based on title and abstract	-1,762 rejected	Fair/good inter-rater agreement on first sift (k statistic test)
Full papers considered for review	311 primary 80 secondary	Each paper is read in full and 80 secondary papers are identified from references
Rejected on full reading	-185 rejected	Papers are rejected on the basis that they do not answer our research questions
Comparison to Catal and Diri's review	2 (which our searches missed)	
Papers accepted for the review	**208 papers**	

Table 3. Paper selection and validation process

3. Assessing the suitability of papers for synthesis

The previous section explained how we initially included papers which both answered our research questions and satisfied our inclusion criteria. In this section we describe how we identified a subset of those papers as suitable from which to extract data and synthesise an overall picture of fault prediction in software engineering. We then describe the extraction and synthesis process.

3.1 The assessment criteria

Our approach to identifying papers suitable for synthesis is motivated by Kitchenham and Charter's [3] notion of a quality check. Our assessment is focused specifically on identifying only papers reporting sufficient information to allow synthesis across studies in terms of answering our research questions. To allow this a basic set of information must be reported in papers. Without this it is difficult to properly understand what has been done in a study and equally difficult to adequately contextualise the findings reported by a study. We have developed and applied a set of criteria focused on ensuring sufficient contextual and methodological information is reported in fault prediction studies. Our criteria are organised in four phases described below.

Phase One: Establishing that the study is a prediction study.

In this SLR it is important that we consider only models which actually do some form of prediction. Some studies which seem to be reporting prediction models actually turn out to be doing very little prediction. Many of these types of studies report correlations between metrics and faults. Such studies only indicate the propensity for building a prediction model. Furthermore, a model is only doing any prediction if it is tested on unseen data (i.e. data that was not used during the training process) [[112]]. To be considered a prediction model it must be trained and tested on different data sets. Table 4 shows the criteria we apply to assess whether a study is actually a prediction study.

Table 4 shows that a study can pass this criterion as long as they have separated their training and testing data. There are many ways in which this separation can be done. Holdout is probably the simplest approach where the original data set is split into two groups comprising: {training set, test set}. The model is developed using the training set and its performance is then assessed on the test set. The weakness in this approach is that results can be biased because of the way the data has been split. A safer approach is often n-fold cross validation, where the data is split into *n* groups {g1..g*n*}. Ten-fold cross validation is very common, where the data is randomly split into ten groups, and ten experiments carried out. For each of these experiments, one of the groups is used as the testing set, and all others combined are used as the training set. Performance is then typically reported as an average across all ten experiments. Stratified cross validation is an improvement to the process, and maintains the proportion of faulty and non-faulty data points being approximately equal to the overall class distribution in each of the *n* bins. Although there are stronger and weaker techniques available to separate training and testing data we have not made a judgment on this and accepted any form of separation in this phase of assessment.

Table 4. Prediction Criteria

Prediction criteria	Criteria definitions	Why the criteria is important
Is a prediction model reported?	The study must report some form of prediction. Not just report a correlation study of the relationship between faults and independent variables.	Such studies provide useful insights into observed patterns of faults but do not present prediction models as such. Nor do they validate their findings using unseen data. We do not therefore take these studies forward for synthesis.
Is the prediction model tested on unseen data?	The prediction model must be developed and tested on different data. So some form of hold-out or cross validation is necessary.	The performance of predictions based only on training data gives us no information on which to judge the performance of how such models generalise to new data. They are not taken forward for synthesis.

Phase Two: Ensuring sufficient contextual information is reported.

We check that basic contextual information is presented by studies to enable appropriate interpretation of findings. A lack of contextual data limits the user's ability to: interpret a model's performance, apply the model appropriately or repeat the study. For example, a model may have been built using legacy systems with many releases over a long time period and has been demonstrated to perform well on these systems. It may not then make sense to rely on this model for a new system where the code is newly developed. This is because the number and type of faults in a system are thought to change as a system evolves [[83]]. If the maturity of the system on which the model was built is not reported, this severely limits a model user's ability to understand the conditions in which the model performed well and to select this model specifically for legacy systems. In this situation the model then may be applied to newly developed systems with disappointing predictive performances.

The contextual criteria we applied are shown in Table 5 and are adapted from the context checklist developed by Peterson and Wohlin [6]. Our context checklist also overlaps with the 40 project characteristics proposed by Zimmermann et al [[208]] as being relevant to understanding a project sufficiently for cross project model building (it was impractical for us to implement all 40 characteristics as none of our included studies report all 40).

Context data is particularly important in this SLR as it is used to answer Research Question 1 and interpret our overall findings on model performance. We only synthesise papers that report all the required context information as listed in Table 5. Note that studies reporting several models based on different data sets can pass the criteria in this phase if sufficient contextual data is reported for one of these models. In this case, data will only be extracted from the paper based on the properly contextualised model.

Table 5. Context Criteria

Contextual criteria	Criteria definitions	Why the criteria is important
Source of data	The source of the system data on which the study is based must be given. For example whether the system data is industrial, open source, NASA, Promise. If NASA/Promise data is used the names of the datasets used must be explicitly given. For studies using data that is in the public domain, the context of which is accessible via the public domain, need not explicitly report all these criteria to pass this phase. However the version studied must be specified to enable access to contextual data.	Different models may perform differently applied to different data sets; for example some models may perform better on OS data than industrial data. It is therefore essential for synthesis that we can establish the source of the data.
Maturity	Some indication of the maturity of the system being studied must be reported. Readers must be able to generally determine whether the system is a mature system with many releases which has been in the field for years or whether it is a relatively newly developed system, or whether the system has yet to be released.	The age of a system has a significant impact on how it behaves. Especially in terms of the faults in the system. Many factors contribute to why the age of the system impacts on faults in the system, including the amount of change the system has undergone. This means that some models are likely to perform better than others on newly developed as opposed to legacy systems. It is therefore essential for that we can establish the maturity of the system(s) on which the model was based, as without this it is difficult to correctly interpret study findings for synthesis.
Size in KLOC	An indication of the size of the system being studied must be given in KLOC. The overall size of the system will suffice, i.e. it is not necessary to give individual sizes of each component of the system being studied, even if only sub-sets of the system are used during prediction. Size indicated by measures other than KLOC are not acceptable (e.g. number of classes) as there are great variations in the KLOC of such other measures.	The size of systems is likely to impact on the behaviour of systems. Consequently, the faults in a system may be different in small as opposed to large systems. This means that it is also likely that different types of models will perform differently on systems of different sizes. It is therefore essential that we can establish the size of the system(s) on which the model was built as without this it is difficult to correctly interpret study findings for synthesis.
Application domain	A general indication of the application domain of the system being studied must be given, e.g. telecoms, customer support etc.	Some models are likely to be domain specific. Different domains apply different development practices and result in different faults. It is therefore important that domain information is given so that this factor can be taken into account when model performance is evaluated. It is therefore essential for that we establish the domain of the system(s) on which the model was built as without this it is difficult to correctly interpret study findings for synthesis.
Programming language	The programming language(s) of the system being studied must be given.	Different languages may result in different faults. In particular it may be that OO languages perform differently from procedural languages. This makes it likely that some models will perform better for some languages. It is therefore essential that we can establish the language of the system(s) on which the model was built as without this it is difficult to correctly interpret study findings for synthesis.

Phase Three: Establishing that sufficient model building information is reported

For a study to be able to help us to answer our research questions it must report its basic model building elements. Without clear information about the independent and dependent variables used as well as the modelling technique, we cannot extract sufficient clear data to allow synthesis. Table 6 describes the criteria we apply.

Table 6. Model Building Criteria

Model building criteria	Criteria definitions	Why the criteria is important
Are the independent variables clearly reported?	The basis on which the model is making predictions must be clear and explicit. For example independent variables (or predictor variables) could include: static code metrics (complexity etc.), code churn metrics, previous faults metrics etc.	In any experimental work which shows the performance of a method it is essential that the independent variables tested are explicitly identified. Without this, confidence in the work is significantly lowered and it is difficult to evaluate the impact of those variables across studies during synthesis.
Is the dependent variable clearly reported?	It must be distinguishable whether studies are predicting faults in terms of whether a module is fault prone or not fault prone (i.e. using a categorical or binary dependent variable) or in terms of the number of faults in a code unit (i.e. using a continuous dependent variable). Some continuous studies additionally report ranked results (i.e. the identification of the faultiest 20% of files).	In any experimental work it is essential that the dependent variables are explicitly identified. Without this, confidence in the work is significantly lowered. Furthermore, the evaluation of fault prediction models is related to whether the dependent variable is categorical or continuous. It is therefore essential for synthesis that this information is clear.
Is the granularity of the dependent variable reported?	The unit of code granularity of predictions must be reported. For example fault predictions in terms of faults per module, per file, per package etc. These terms may be used differently by different authors, e.g. 'module' is often used to mean different code units by different authors. Studies must indicate what code unit is being used within the terms used, i.e. if the term 'module' is used, readers must be able to work out what unit of code is being defined as a module.	It is difficult to directly compare the performance of one model reporting faults per file to another model reporting faults per method. Furthermore it may be that studies reporting faults at a file level are more able to perform well than studies reporting at a method level. The granularity of the dependent variable must therefore be taken into account during synthesis and so an indication of the unit of fault granularity must be reported.
Is the modelling technique used reported?	It must be clear what general modelling technique is being used, e.g. machine learning, regression etc. A model must be presented, not just a tool based on a model that is not included in the paper.	Different modelling techniques are likely to perform differently in different circumstances. A study must report the modelling technique used, as we cannot examine the impact of method on performance without this information.

Phase Four: Checking the model building data

The data used is fundamental to the reliability of the models. Table 7 presents the criteria we apply to ensure that studies report basic information on the data they used.

Table 7. Data Criteria

Data criteria	Criteria definitions	Why the criteria is important
Is the fault data acquisition process described?	The process by which the fault data was obtained must be described. A high level indication of this will suffice, e.g. obtained from the CVS records. However at least an overview of how the data was obtained must be provided. If the data has been obtained from a third party (e.g. NASA), some indication of how that data was obtained by that third party must be given or referenced or available in the public domain.	In order to have confidence in the data on which a model was built it is necessary to have some information on how the data was collected. Data is fundamental to the quality of the models built and the collection process has a huge impact on the quality of that data. It is not possible to have any confidence in a study where the data seems to have come from thin air. Collecting software engineering data of any sort is difficult and error-prone. We do not synthesise papers which give no indication of how the data was collected.
Is the independent variable data acquisition process described?	Some indication of how the independent variable data (e.g. static code data) was obtained should be given. For example the static analysis tools used should be reported or the process by which code churn data identified should be described. This does not need to be at a great level of detail, just an indication given. If the data has been obtained from a third party (e.g. NASA), some indication of how that data was obtained by that third party must be given or referenced or available in the public domain.	As above.
For categorical studies, has the number of faulty versus non-faulty units on which the model has been trained and tested on been reported?	The balance of faulty versus non-faulty units used for training and testing must be reported. For studies using open source systems it is not essential to report this (though preferable) as this data should be possible to identify from the public data source.	The balance of faulty versus non-faulty units used for training and testing can affect the reliability of some performance measures (see Appendix F). It is essential that class distributions are reported (i.e. number of faulty and number of non-faulty units in the data used). This makes it possible to appropriately interpret performances reported using these measures. We use this information for our synthesis of categorical studies. As where precision and recall are not reported by such studies we re-compute an approximation of it. The faulty/non-faulty balance of data is often needed in this calculation.

In addition to the criteria we applied in Phases One to Four, we also developed more stringent criteria that we did not apply. These additional criteria relate to the quality of the data used and the way in which predictive performance is measured. Although we initially intended to apply these additional criteria, this was not tenable because the area is not sufficiently mature. Applying these criteria would probably result in only a handful of studies being synthesised. We include these criteria in Appendix C as they identify further important criteria that future researchers should consider when building models.

3.2 Applying the assessment criteria

Our criteria have been applied to our included set of 208 fault prediction studies. This identified a subset of 35 finally included studies from which we extracted data and on which our synthesis is based. The initial set of 208 included papers was divided between the five authors. Each paper was assessed by two authors independently (with each author being paired with at least three other authors). Each author applied the assessment criteria to between 70 and 80 papers. Any disagreements on the assessment outcome of a paper were discussed between the two authors and, where possible, agreement established between them. Agreement could not be reached by the two authors in 15 cases. These papers were then given to another member of the author team for moderation. The moderator made a final decision on the assessment outcome of that paper.

We applied our four phase assessment to all 208 included studies. The phases are applied sequentially. If a study does not satisfy all of the criteria in a phase the evaluation is stopped and no subsequent phases are applied to the study. This is to improve the efficiency of the process as there is no point in assessing subsequent criteria if the study has already failed the assessment. It does have the limitation that we did not collect information on how a paper performed in relation to all assessment criteria. So if a paper fails Phase One we have no information on how that paper would have performed in Phase Four.

This assessment process was piloted four times. Each pilot involved three of the authors applying the assessment to 10 included papers. The assessment process was refined as a result of each pilot.

We developed our own MySQL database system to manage this SLR. The system recorded full reference details and references to pdf's for all papers we identified as needing to be read in full. The system maintained the status of those papers as well as providing an on-line process to support our assessments of 208 papers. The system collected data from all authors performing assessments. It also provided a moderation process to facilitate identifying and resolving disagreements between pairs of assessors. The system eased the administration of the assessment process and the analysis of assessment outcomes. All data that was extracted from the 35 papers which passed the assessment is also recorded on our system. An overview of the system is available from [7] and full details are available by emailing the third author.

3.3 Extracting data from papers

Data addressing our three research questions was extracted from each of the 35 finally included studies which passed all assessment criteria. Our aim was to gather data that would allow us to analyse predictive performance within individual studies and across all studies. To facilitate this, three sets of data were extracted from each study:

1. **Context data.** Data showing the context of each study was extracted by one of the authors. This data gives the context in terms of: the source of data studied, as well as the maturity, size, application area and programming language of the system(s) studied.

2. **Qualitative data.** Data related to our research questions was extracted from the findings and conclusions of each study. This was in terms of what the papers reported rather than on our own interpretation of their study. This data supplemented our quantitative data to generate a rich picture of results within individual studies.

   Two authors extracted qualitative data from all 35 studies. Each author extracted data independently and compared their findings to those of the other author. Disagreements and omissions were discussed within the pair and a final set of data agreed upon by them.

3. **Quantitative data.** Predictive performance data was extracted for every individual model (or model variant) reported in a study. The performance data we extracted varied according to whether the study reported their results via categorical or continuous dependent variables. Some studies reported both categorical and continuous results. We extracted only one of these sets of results depending on the way in which the majority of results were presented by those studies. The following is an overview of how we extracted data from categorical and continuous studies.

   Categorical studies. There are 22 studies reporting categorical dependent variables. Categorical studies report their results in terms of predicting whether a code unit is likely to be fault prone or not fault prone. Where possible we report the predictive performance of these studies using precision, recall and f-measure (as many studies report both precision and recall, from which an f-measure can be calculated). F-measure generally gives a good overall picture of predictive performance[3]. We used these three measures to compare results across studies, and where necessary calculate and derive these measures from those reported (Appendix E explains how we did this conversion and shows how we calculated f-measure). Standardising on the

[3] Using precision alone has some limitations as discussed in [18]. Menzies et al [19] also show that the values of precision vary greatly when used with models using different data sets. However reporting precision in combination with recall mitigates some of these problems.

performance measures reported allows comparison of predictive performances across studies. Lessmann et al [[97]] recommend the use of consistent performance measures for cross study comparison; in particular, they recommend use of Area Under the Curve (AUC). We also extract AUC where studies report this.

We present the performance of categorical models in box plots. Box plots are useful for graphically showing the differences between populations. They are useful for our results as they make no assumptions about the distribution of the data presented. These box plots present the precision, recall and f-measure of studies according to a range of model factors. These factors are related to the research questions presented at the beginning of Section Two and, an example is a box plot showing model performance relative to the modelling technique used. Appendix G gives more information on the format of our boxplots.

Continuous studies. There are 13 studies reporting continuous dependent variables. These studies report their results in terms of the number of faults predicted in a unit of code. It was not possible to convert the data presented in these studies into a common comparative measure; we report the individual measures that they use. Most measures reported by continuous studies are based on reporting an error measure (e.g. Mean Standard Error (MSE)), or else measures of difference between expected and observed results (e.g. Chi Square). Some continuous studies report their results in ranking form (e.g. top 20% of faulty units). We extract the performance of models using whatever measure each study used.

Two authors extracted quantitative data from all 35 studies. A pair approach was taken to extracting this data since it was a complex and detailed task. This meant that the pair of authors sat together identifying and extracting data from the same paper simultaneously.

3.4 Synthesising data across studies

Synthesising findings across studies is notoriously difficult and many software engineering SLRs have been shown to present no synthesis [8]. In this paper, we have also found synthesising across a set of disparate studies very challenging. We extracted both quantitative and qualitative data from studies. We intended to meta-analyse our quantitative data across studies by combining precision and recall performance data. However the studies are highly disparate in terms of both context and models. Meta-analysing this quantitative data would generate unsafe results. Such a meta-analysis would suffer from many of the limitations in SLRs published in other disciplines [9].

We combined our qualitative and quantitative data to generate a rich picture of fault prediction. We did this by organising our data into themes based around our three research questions (i.e. context, independent variables and modelling techniques). We then combined the data on each theme to answer our research questions. This synthesis is presented in Section Seven.

4 Threats to validity

Searches: We do not include the term 'quality' in our search terms as this would have resulted in the examination of a far wider range of irrelevant papers. This term generates a high number of false positive results. We might have missed some papers that use the term 'quality' as a synonym for 'defect' or 'fault', etc. However we missed only two papers that Catal and Diri's [2] searches found using the term 'quality'. This gives us confidence that we have missed very few papers. We also omitted the term 'failure' from our search string as this generated papers predominately reporting on studies of software reliability in terms of safety critical systems. Such studies of reliability usually examine the dynamic behaviour of the system and seldom look at the prediction of static code faults which is the focus of this review.

We apply our search terms to only the title of papers. We may miss studies that do not use these terms in the title. Since we extend our searches to include papers cited in the included papers, as well as key conferences, individual journals and key authors, we are confident that the vast majority of key papers have been included.

Studies included for synthesis: The 35 studies which passed our assessment criteria may still have limitations that make their results unreliable. In the first place, the data on which these models are built might be problematic as we did not insist that studies report data cleaning or attribute selection. As a consequence, the data on which some studies is based may be unreliable. Nor did we apply any performance measure-based criteria. So some studies may be reporting unsafe predictive performances. This is a particular risk in regard to how studies have accounted for using imbalanced training data. This risk is mitigated in the categorical studies where we are able to report precision, recall and f-measure.

It is also possible that we have missed studies which should have been included in the set of 35 from which we extracted data. Some studies may have satisfied our assessment criteria but either failed to report what they did or did not report it in sufficient detail for us to be confident that they should pass the criteria. Similarly we may have missed the reporting of a detail and a paper that should have passed a criterion did not. These risks are mitigated by two authors independently assessing every study.

The box plots: The boxplots we present set performance against individual model factors (e.g. modelling technique used). This is a simplistic analysis, as a number of interacting factors are likely to underpin the performance of a model. For example, the technique used in combination with the data set and the independent variables are likely to be more important than any one factor alone. Furthermore methodological issues are also likely to impact on performance, for example whether feature selection has been used. Our box plots only present possible indicators of factors that should be investigated within the overall context of a model. More sophisticated analysis of a larger data set is needed to investigate the factors that influence the performance of a model.

Our box plots do not indicate the direction of any relationship between model performance and particular model factors. For example we do not investigate whether a particular modelling technique performs well because it was used in a good model or whether a model performs well because it used a particular modelling technique. This is also important work for the future. In addition, some studies contribute data from many models to one box plot whereas other studies contribute data from only one model. This may skew the results. We do not calculate the statistical significance of any differences observed in the box plots. This is because the data contained within them is not normally distributed and the individual points represent averages from different sizes of population.

5 Results of our assessment

In this section, we present the results from applying our assessment criteria (detailed in Tables 4, 5, 6 and 7) to establish whether or not a paper reports sufficient contextual and methodological detail to be synthesized. The assessment outcome for each study is shown at the end of its reference in the list of included studies.

Table 8: Results of applying assessment criteria

Number of papers passed	Number of papers failed					
	Phase 1: Prediction	Phase 2: Context	Phase 3: Model	Phase 4: Data	Other reasons	Total failed
35	41	114	2	13	3	173

Table 8 shows that only 35 of our initially included 208 studies passed all assessment criteria[4]. Of these 35 finally included studies, three are relatively short ([[116]], [[110]] and [[164]]). This means that it is possible to report necessary contextual and methodological detail concisely without a significant overhead in paper length. Table 8 also shows that 41 papers failed at phase 1 of the assessment because they did not report prediction models as such. This includes studies that only present correlation studies or models that were not tested on data (e.g. hold-out data) unseen during training. This is an important finding as it suggests that a relatively high number of papers reporting fault prediction are not really doing any prediction.

Table 8 also shows that 13 studies provided insufficient information about their data. Without this it is difficult to establish the reliability of the data on which the model is based. Table 8 also shows that a very high number of studies (114) reported insufficient information on the context of their study. This makes it difficult to interpret the results reported in these studies and to select an appropriate model for a particular context. Several studies passing all of our criteria anonymised their contextual data see, for example [[109]] and [[110]]. Although these studies gave full contextual details of the systems they used, the results associated with each were anonymised. This meant that it was impossible to relate specific fault information to specific systems. While a degree of commercial confidentiality was maintained, this limited our ability to analyse the performance of these models.

Of the 114 studies which did not report sufficient context information, 58 were based on NASA data (located in MDP or PROMISE). This is because we could find no information about the maturity of the systems on which the NASA data is based. Maturity information is not given in either the MDP or PROMISE repository documentation and no included paper provided any maturity information. Turham et al [10] report that the NASA data is from numerous NASA contractors for an array of projects with a wide range of reuse. This suggests that a range of maturities might also be represented in these datasets. No clear insight is given into whether particular data sets are based on systems developed from untested, newly released or legacy code based on many releases. The only three studies using NASA data which passed the context phase of the assessment were those also using other data sets for which full context data is available (the NASA based models were not extracted from these studies). Whether a study uses NASA data (sourced from MDP or PROMISE) is shown at the end of its reference in the list of included studies.

Table 8 also shows that three studies failed the assessment due to the 'other' reasons reported in Table 9.

Table 9: Issues with the measurement of performance

Paper number	Performance measurement issues
[[141]]	The model was optimised on the test set which will result in inflated performance as shown by the 100% accuracy result.
[[52]]	The paper does not report the error rate of any of the models it presents. Only the difference between the error rates reported by each model reported is given. However the error rate could be very high for all models, but the difference between each model could be very small giving the impression that the models are all working well.
[[127]]	There appears to be a calculation error in the performance measures reported.

[4] These papers are: [[8]], [[11]], [10]], [[37]], [[51]], [[83]], [[92]], [[98]], [[116]], [[117]], [[122]], [[133]], [[154]], [[31]], [[73]], [[110]], [[190]], [[86]], [[21]], [[29]], [[32]], [[56]], [[69]], [[76]], [[109]], [[118]], [[120]], [[135]], [[164]], [[18]], [[74]], [[163]], [[9]], [[160]], [[203]].

6 Results extracted from papers

In this section we present the results we extracted from the 35 papers that passed all of our assessment criteria. The full set of data extracted from those papers is contained in our on-line appendix (https://bugcatcher.stca.herts.ac.uk/slr2011/). This on-line appendix consists of the following:

1. **Context of Study Table**. For each of the 35 studies, the context of the study is given in terms of: the aim of the study together with details of the system(s) used in the study (the application area(s), the system(s), maturity and size(s)).

2. **Categorical Models Table**. For each study reporting categorical results, each model is described in terms of the: independent variable(s), the granularity of the dependent variable, the modelling technique(s) and the data set(s) used. This table also reports the performances of each model using precision, recall, f-Measure and (where given by studies) AUC. Some studies present many models or model variants, all of which are reported in this table.

3. **Continuous Models Table**. For each study reporting continuous results (including those reporting ranking results) the same information describing their model(s) is presented as for categorical models. However the performance of each continuous model is reported in terms of either: the error measure, the measure of variance or the ranked results (as reported by a study).

   Models which report both categorical and continuous results appear only in one table. The choice of categorical or continuous table is based on the way in which the majority of results are presented by those studies.

4. **Qualitative Data Table.** For each study a short summary of the main findings reported by studies is presented.

The remainder of this section contains box plots illustrating the performance of the models in relation to various model factors (e.g. modelling technique used, independent variable used etc.). These factors are related to the research questions that we posed at the beginning of Section Two. The box plots in this section set performance against individual model factors (e.g. modelling technique used). This is a simplistic analysis, as a range of interacting factors are likely to underpin the performance of a model. However our results indicate areas of promising future research.

The box plots represent models reporting only categorical results for which precision, recall and f-measure were either reported or could be calculated by us. Such models are reported in 18 of the 22 categorical studies (of the remaining four, three report AUC). We are unable to present box plots for the 13 studies using continuous data as there the measures used are not comparable or convertible to comparable measures.

Each box plot includes data only where at least three models have used a particular factor (e.g. a particular independent variable like LOC). This means that the numbers (*n*) at the top of the box plots will not add up to the same number on every plot, as factors used in less than three studies will not appear; the total of *n*'s will therefore vary from one box plot to the next. The box plots contain performance data based on precision, recall and f-measure. This is for all categorical models and model variants presented by each study (203 models or model variants). Some studies present many model variants while others present only one model. We also created box plots of only the best results from each study. These box plots did not change the pattern of good performances but only presented limited information about poor performances. For that reason, we do not include these 'best only' box plots. Appendix G describes how to interpret the box plots presented.

6.1 Performances of models reported in individual Studies

Figure 1 is a box plot of the performances of all the models reported by each of the 18 categorical papers (full details of which can be found in the on-line appendix). For each individual paper f-measure, precision and recall is reported. Figure 1 shows that studies report on many models or variants of models, some with a wide range of performances (the details of these can be found in the Models Table in the on-line Appendix (https://bugcatcher.stca.herts.ac.uk/slr2011/)). For example, Schroter et al [[154]] present 20 model variants with a wide range of precision, recall and f-measure. Many of these variants are not particularly competitive; the most competitive models that Schroter et al [[154]] report are based on training the model on only the faultiest parts of the system. This is a promising training technique and a similar technique has also been reported to be successful by Zhang et al [[200]]. Bird et al [[18]] report 28 model variants with a much smaller range of performances, all of which are fairly competitive. Figure 1 also shows the performance trade-offs in terms of precision and recall made by some models. For example, Bird et al [[18]] report consistent precision and recall, whereas Nagwani and Verma [[118]] and Shivaji et al [[164]] report performances where precision is much higher than recall.

Figure 1. Performances of the models reported in each of the categorical studies

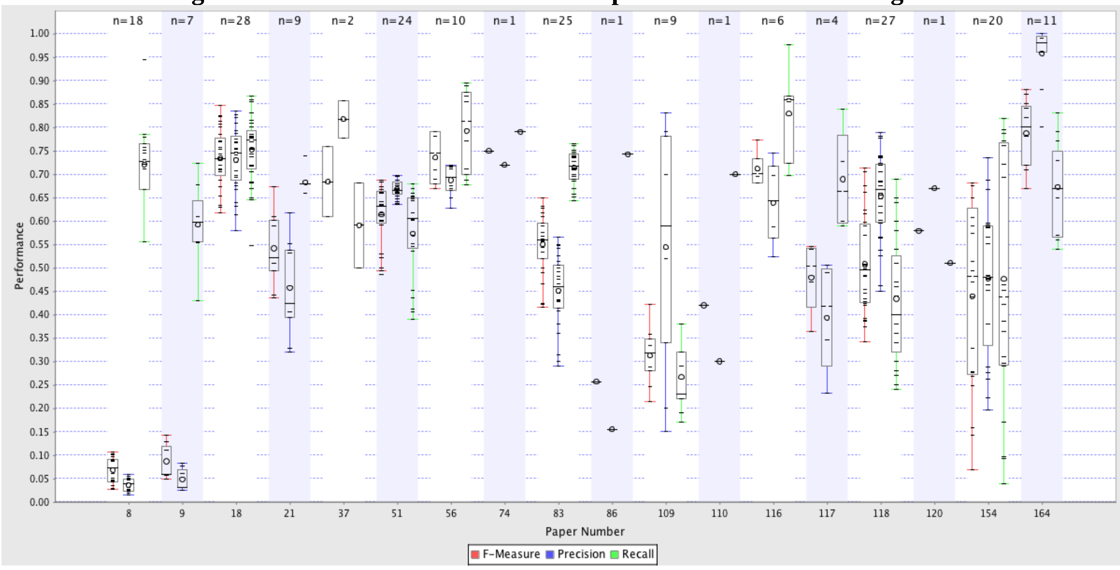


Figure 1 also shows that some models seem to be performing better than others. The models reported by Shivaji et al [[164]] are performing extremely competitively (especially in terms of precision). Naïve Bayes is the technique used by Shivaji et al [[164]]; Naïve Bayes has performed generally well across the studies (Shivaji et al [[164]] further optimised their Naïve Bayes technique). Modelling technique is reported in Figure 8. Shivaji et al [[164]] used a wide range of independent variables; these were also then optimised using feature selection. In addition, they evaluated which performance measure to use when training the model. The data used in Shivaji et al [[164]] also contains a good proportion of faults which means that the training data is fairly balanced. This may improve performance by providing many examples of faults from which the modelling technique can train. Clearly there are many good aspects of this study that mean it is likely to produce models which perform well. On the other hand the performance of Arisholm et al's models ([[8]] and [[9]]) are low in terms of precision but competitive in terms of recall. Both studies are different but use the same data sets. This low precision is reportedly because of the sampling method used to address the imbalance of the data used. Though the data sets used are also small relative to those used in other studies (148KLOC). Arisholm et al's studies ([[8]] and [[9]]) are interesting as they also report many good modelling practices and in some ways are exemplary studies. But they demonstrate how the data used can impact significantly on the performance of a model. It is also essential that both high and low performances be reported, as it is only by identifying these that our overall understanding of fault prediction will improve. The boxplots in the rest of this section explore in more detail aspects of models that may underpin these performance variations. Because the performances of Arisholm et al's models ([[8]] and [[9]]) are very different from those of the other studies we have removed them from the rest of the box plots. We have treated them as outliers which would skew the results we report in other box plots.

6.2 Performances in relation to context factors

Figure 2 shows the data sets used in the studies. It shows that 108 models reported in the studies are based on data from Eclipse. Eclipse is very well studied, probably because the fault data is easy to access and its utility has been well proven in previous studies. In addition, data already extracted from Eclipse is available from Saarland University (http://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/) and PROMISE (http://promisedata.org/). Figure 2 shows that there is a wide variation in model performance using Eclipse. Figure 2 also suggests that it may be more difficult to build models for some systems than for others. For example, the models built for embedded telecoms systems are not particularly competitive. This may be because such systems have a different profile of faults with fewer post-delivery faults relative to other systems. Developers of such systems normally prioritise reducing post-delivery faults as their embedded context makes fixing them comparatively expensive [[83]].

Figure 2. Data used in models

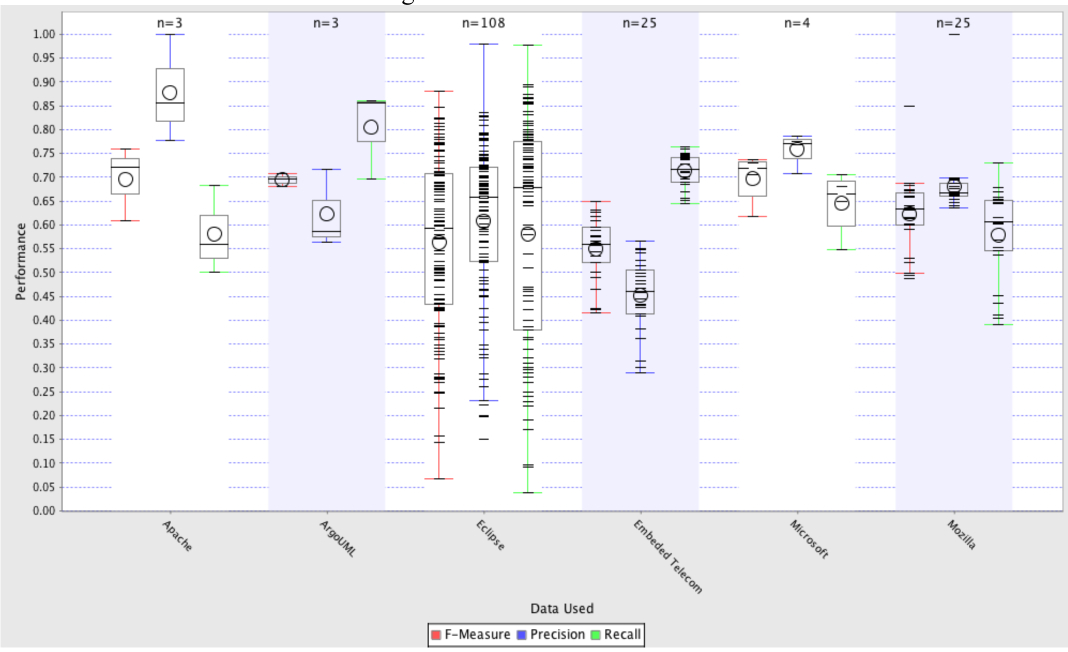


Figure 3 shows how models have performed relative to the size of systems on which they are based. Figure 3 is quite a difficult figure to interpret; however it suggests that as the size of a system increases, model performance seems to improve. Figure 3 shows this via the KLOC increase in releases of Eclipse used by studies (Eclipse versions 2.0, 2.1, 3.0, 3.1, 3.2 and 3.3 are represented on Figure 3 by KLOC: 797, 946, 1255, 1537, 1823 and 1833). As these Eclipse releases increase in size, a weak trend of improved predictive performance can be observed. This makes sense as models are likely to perform better given more data.

Figure 3. The size of the data sets used

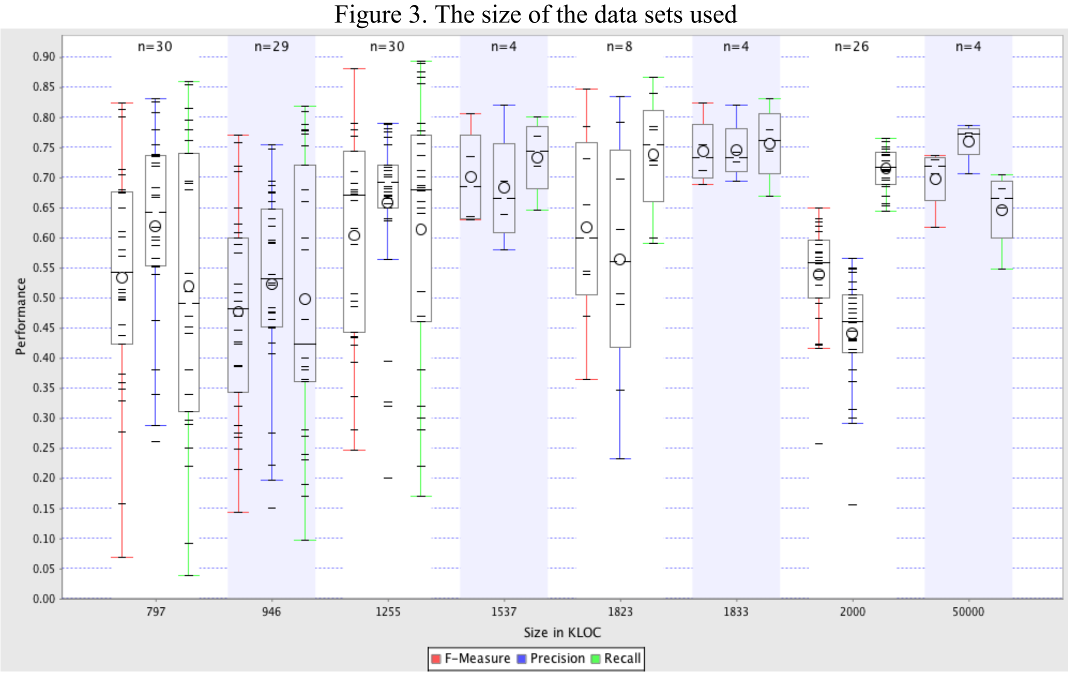


Figure 4 shows the maturity of systems used by studies relative to the performance of models. The Context Table in the on-line appendix shows how systems have been categorised in terms of their maturity. Figure 4 shows that no immature systems are used by more than two models in this set of studies (i.e. where n ≥ 3)[5]. There seems to be little difference between the performance of models

[5] An exception to this is found in studies [[11]], [[133]] where immature systems are used with promising performances reported (see online appendix for full details).

using mature or very mature systems. This suggests that the maturity of systems may not matter to predictive performance[6]. This finding may be linked to the finding we report on size. It may be that what was previously believed about the importance of maturity was actually about size i.e. maturity is a surrogate for size. However we do not have enough data to draw conclusions here, and the data we analyse contains no studies using immature systems. More research is needed to test for possible association between maturity and size, and whether data extracted from immature systems can be used as a basis for reliable fault prediction.

Figure 4. The maturity of the systems used

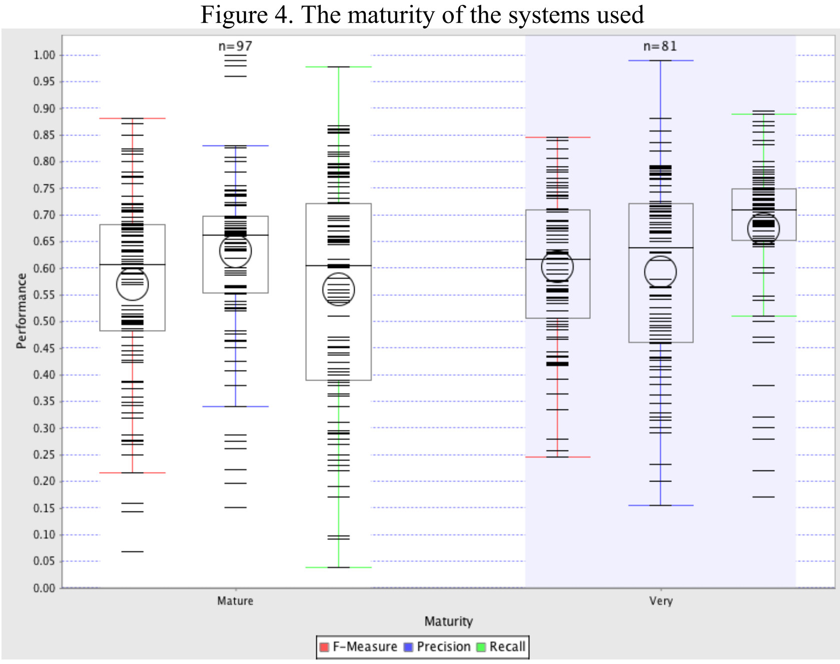


Figure 5 shows the language used in the systems studied in relation to the performance of models. We present only studies reporting the use of either Java or C/C++. There are several single studies using other languages which we do not report. Figure 5 suggests that model performance is not related to the language used in systems.

Figure 5. The language used

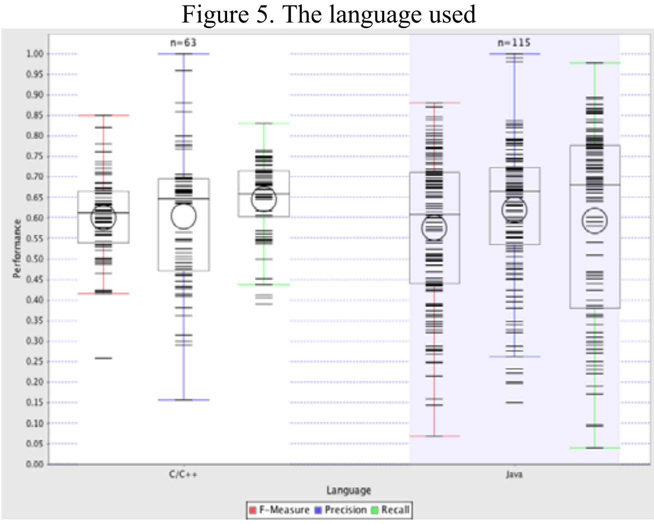
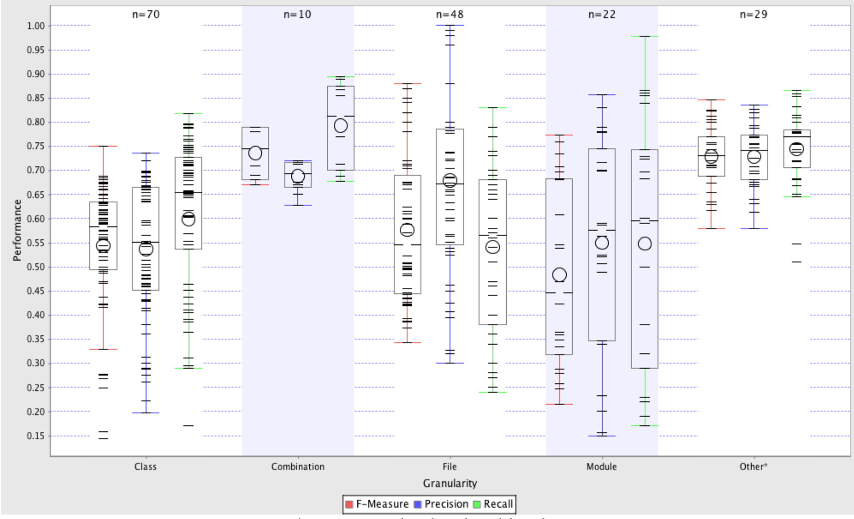


Figure 6 shows model performance relative to the granularity of dependent variables (e.g. whether fault prediction is at the class or file level). It shows no clear relationship between granularity and performance. It does not seem to be the case that increased granularity is clearly related to improved performance. Models reporting at ‘other’ levels of granularity seem to be performing most consistently. These tend to be high levels of granularity defined specifically by individual studies (e.g. Nagappan et al [[120]]).

6 This may mean that it is not important to report maturity when studies describe their context (many more studies would have passes our assessment had that been the case). However much more data on maturity is needed before firm conclusions can be drawn.

Figure 6. The granularity of the results

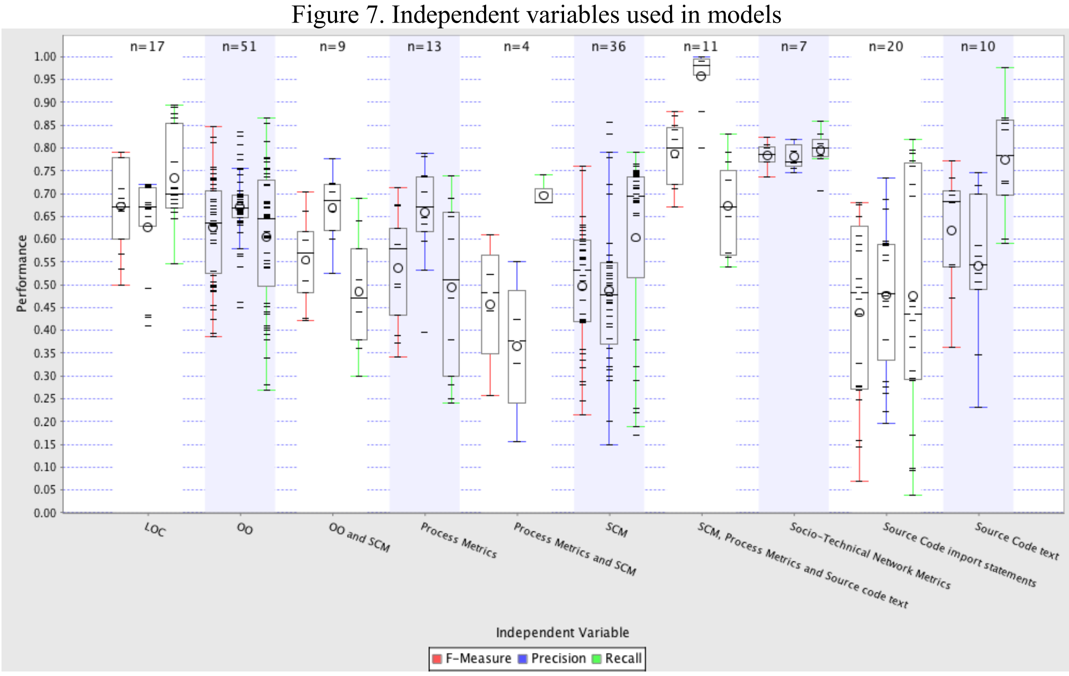


* For example plug-ins, binaries

6.3 Performance in relation to independent variables

Figure 7 shows model performance in relation to the independent variables used. The Categorical Models Table in the on-line appendix shows how independent variables as expressed by individual studies have been categorised in relation to the labels used in Figure 7. It shows that there is variation in perfomance between models using different independent variables. Models using a wide combination of metrics seem to be performing well. For example, models using a combination of static code metrics (scm), process metrics and source code text seem to be performing best overall (e.g. Shivaji et al [[164]]). Similarly Bird et al's study [[18]] which uses a wide combination of socio-technical metrics (code dependency data together with change data and developer data) also performs well (though the results from Bird et al's study [[18]] are reported at a high level of granularity). Process metrics (i.e. metrics based on changes logged in repositories) have not performed as well as expected. OO metrics seem to have been used in studies which perform better than studies based only on other static code metrics (e.g. complexity based metrics). Models using only LOC data seem to have performed competitively compared to models using other independent variables. Indeed of these models using only metrics based on static features of the code (OO or SCM), LOC seems as good as any other metric to use. The use of source code text seems related to good performance. Mizuno et al's studies ([[116]] and [[117]]) have used only source code text within a novel spam filtering approach to relatively good effect.

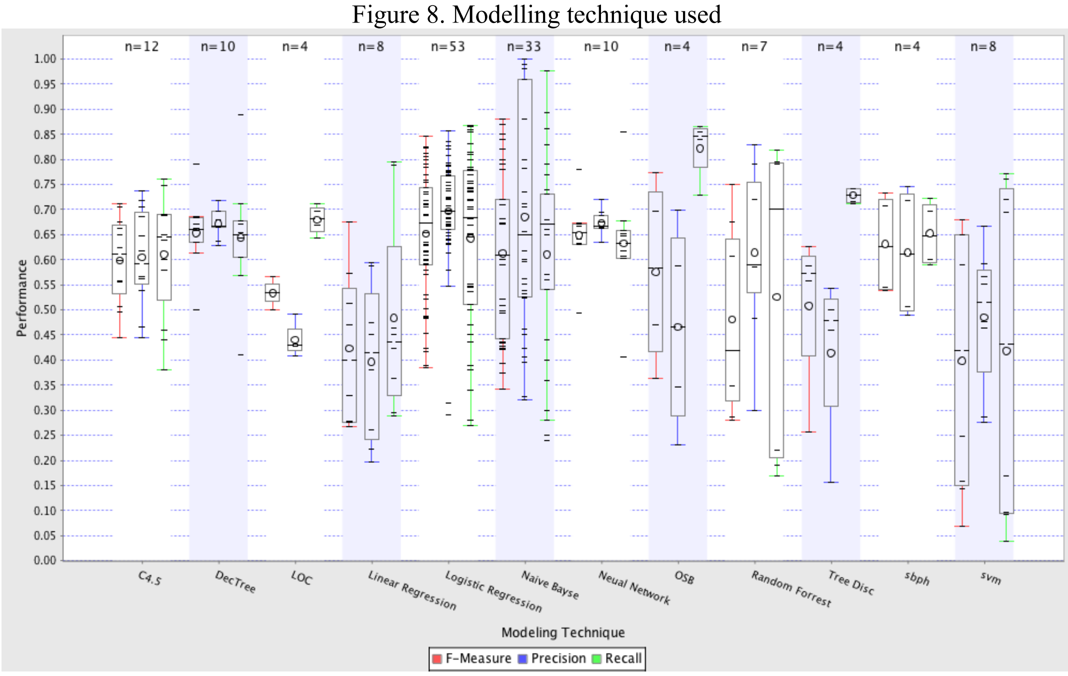
Figure 7. Independent variables used in models



6.4 Performance in relation to modelling technique

Figure 8 shows model performance in relation to the modelling techniques used. Models based on Naïve Bayes seem to be performing well overall. Naïve Bayes is a well understood technique that is in common use. Similarly models using Logistic Regression also seem to be performing well. Models using Linear Regression perform not so well, though this technique assumes that there is a linear relationship between the variables. Studies using Random Forests have not performed as well as might be expected (many studies using NASA data use Random Forests and report good performances [[97]]). Figure 8 also shows that SVM (Support Vector Machine) techniques do not seem to be related to models performing well. Furthermore, there is a wide range of low performances using SVMs. This may be because SVMs are difficult to tune and the default Weka settings are not optimal. It is also interesting to note that Arisholm et al's models ([[8]] and [[9]]) used the C4.5 technique (as previously explained these are not shown as their relatively poor results skew the data presented). C4.5 is thought to struggle with imbalanced data [11] and this may explain this poor performance.

Figure 8. Modelling technique used



7 Synthesis of results

In this section we answer our research questions by synthesising the qualitative and quantitative data we have collected. The qualitative data consists of the main findings reported by each of the individual 35 finally included studies (presented in the Qualitative Data Table in our on-line appendix). The quantitative data consists of the predictive performance of the individual models reported in the 35 studies (summarised in the Categorical and Continuous Models Tables in our on-line appendix). The quantitative data also consists of the detailed predictive performance data from 18 studies (203 models or model variants) comparing performance across models (reported in Section Six). This combination of data addresses model performance across studies and within individual studies. This allows us to discuss model performance in two ways. First, we discuss performance within individual studies to identify the main influences on model performance reported within a study. Second we compare model performances across the models reported in 18 studies. This is an important approach to discussing fault prediction models. Most studies report at least one model which performs 'well'. Though individual studies usually only compare performance within the set of models they present to identify their best model. We are able to then compare the performance of the models which perform well within a study, across other studies. This allows us to report how well these models perform across studies.

7.1 Answering our research questions

RQ1: How does context affect fault prediction?

Analysing model performance across the 18 studies in detail, suggests that some context variables may influence the reliability of model prediction. Our results provide some evidence to suggest that predictive performance improves as systems get larger. This is suggested by the many models built for the Eclipse system. As Eclipse increases in size the performance of models seems to improve. This makes some sense as models are likely to perform better with more data. We could find no evidence that this improved performance was based on the maturing of systems. It may be that size influences predictive performance more than system maturity. However our data set is relatively small and although we analysed 203 models (or model variants) very few were based on immature systems. Our results also suggest that some applications may be less likely to produce reliable prediction models. For example, the many models built for embedded telecoms applications generally performed less well relative to other applications. Our results also show that many models have been built using Eclipse data. This corpus of knowledge on Eclipse provides a good opportunity for future researchers to meta-analyse across a controlled context.

The conventional wisdom is that context determines how transferrable a model is to other systems. Despite this, none of the 35 finally included studies directly investigate the impact on model performance of specific context variables such as system size, maturity, application area or programming language. One exception is Cruz et al [[29]] who demonstrate that transforming project data can make a model more comparable to other projects.

Many of the 35 finally included studies individually test how well their model performs when transferred to other contexts (releases, systems, application areas, data sources or companies). Few of these studies directly investigate the contextual factors influencing the transferability of the model. Findings reported from individual studies on model transferability are varied. Most studies report that models perform poorly when transferred. In fact Bell et al [[11]] report that models could not be applied to other systems. Denaroro and Pezzè [[37]] reported good predictive performance only across homogenous applications. Nagappan et al [[122]] report that different subsets of complexity metrics relate to faults in different projects and that no single set of metrics fits all projects. Nagappan et al [[122]] conclude that models are only accurate when trained on the same or similar systems. However other studies report more promising transferability. Weyuka et al [[190]] report good performance when models are transferred between releases of systems and between other systems. However Shatnawi and Li [[160]] report that the performance of models declines when applied to later releases of a system. Shatnawi and Li [[160]] conclude that a different set of metrics should be used in models used for later releases.

The context of models has not been studied extensively in the set of studies we analysed. Although every model has been developed and tested within particular contexts, the impact of that context on model performance is scarcely studied directly. This is a significant gap in current knowledge as it means we currently do not know what context factors influence how well a model will transfer to other systems. It is therefore imperative that studies at least report their context since, in the future, this will enable a meta-analysis of the role context plays in predictive performance.

RQ2: Which independent variables should be included in fault prediction models?

Many different independent variables have been used in the 35 finally included studies. These mainly fall into process (e.g. previous change and fault data) and product (e.g. static code data) metrics as well as metrics relating to developers. In addition, some studies have used the text of the source code itself as the independent variables (e.g. Mizuno et al [[116]] and [[117]]).

Model performance across the 18 studies we analysed in detail suggests that the spam filtering technique, based on source code, used by Mizuno et al ([[116]] and [[117]]) performs relatively well. On the other hand models using only static code metrics (typically complexity-based) perform relatively poorly. Model performance does not seem to be improved by combining these metrics with OO metrics. Models seem to perform better using only OO metrics rather than only source code metrics. However models using only LOC seem to perform just as well as those using only OO metrics and better than those models only using source code metrics. Within individual studies, Zhou et al [[203]] report that LOC data performs well. Ostrand et al [[133]] report that there was some value in LOC data and Zhang [[56]] reports LOC to be a useful early general indicator of fault-proneness. Zhou et al [[203]] report that LOC performs better than all but one of the Chidamber and Kemerer metrics (Weighted Methods per Class). Within other individual studies LOC data was reported to have poor predictive power and be out-performed by other metrics (e.g. Bell et al [[11]]). Overall LOC seem to be generally useful in fault prediction.

Model performance across the 18 studies that we analysed suggests that the use of process data is not particularly related to good predictive performance. However looking at the findings from individual studies, several report that process data, in the form of previous history data, performs well (e.g. [[163]], [[120]]). D'Ambros et al [[31]] specifically report that previous bug reports are the best predictors. More sophisticated process measures have also been reported to perform well. In particular Nagappan et al [[120]] introduce ‘change burst’ metrics which demonstrate good predictive performance (however these models perform only moderately when we compared them against models from other studies).

The few studies using developer information in models report conflicting results. Ostrand et al [[135]] report that the addition of developer information does not improve predictive performance much. Bird et al [[18]] report better performances when developer information is used as an element within a socio-technical network of variables. This study also performs well in our detailed comparison of performances (Bird et al [[18]] report results at a high level of granularity and so might be expected to perform better).

The models which perform best in our analysis of 18 studies seem to use a combined range of independent variables. For example Shivaji et al [[164]] use process-based and SCM-based metrics together with source code. Bird et al [[18]] combine a range of metrics. The use of feature selection on sets of independent variables seems to improve the performance of models (e.g. [[164]], [[76]], [[18]]). Optimised sets of metrics using, for example, feature selection, make sense.

RQ3: Which modelling techniques perform best when used in fault prediction?

While many included studies individually report the comparative performance of the modelling techniques they have used, no clear consensus on which perform best emerges when individual studies are looked at separately. Mizuno et al [[117]] report that, of the techniques they studied, Orthogonal Sparse Bigrams Markov models (OSB) are best suited to fault prediction. Bibi et al [[4]] report that Regression via Classification (RvC) works well. Khoshgoftaar et al [[86]] report that modules whose fault proness is predicted as uncertain, can be effectively classified using the TreeDisc (TD) technique. Khoshgoftaar and Seliya [[83]] also report that Case Based Reasoning (CBR) does not predict well with C4.5 also performing poorly. Arisholm et al [[9]] report that their comprehensive performance comparison revealed no predictive differences between the eight modelling techniques they investigated.

A clearer picture seems to emerge from our detailed analysis of model performance across the 18 studies. Our findings suggest that performance may actually be linked to the modelling technique used. Overall our comparative analysis suggests that studies using Support Vector Machine (SVM) techniques perform less well. These may be underperforming as they require parameter optimization (something rarely carried out in fault prediction studies) for best performance [12]. Where SVM's have been used in other prediction domains, and may be better understood, they have performed well [13]. Models based on C4.5 seem to underperform if they use imbalanced data (e.g. Arisholm et al [[8]] and [[9]]), as the technique seems to be sensitive to this. Our comparative analysis also suggests that the models performing comparatively well are relatively simple techniques that are easy to use and well understood. Naïve Bayes and Logistic regression, in particular, seem to be the techniques used in models that are performing relatively well. Models seem to have performed best where the right technique has been selected for the right set of data. And these techniques have been tuned to the model (e.g. Shivaji et al [[164]]), rather than relying on the default Weka parameters.

8 Methodological issues in fault prediction

The methodology used to develop, train, test and measure the performance of fault prediction models is complex. However the efficacy of the methodology used underpins the confidence which we can have in a model. It is essential that models use and report a rigorous methodology. Without this the maturity of fault prediction in software engineering will be low. We identify methodological problems in existing studies so that future researchers can improve on these.

Throughout this SLR methodological issues in the published studies came to light. During our assessment of the 208 initially included studies and the extraction of data from the 35 finally included studies methodological weaknesses emerged. In this section we discuss the most significant of these methodological weaknesses. These generally relate to the quality of data used to build models and the approach taken to measure the predictive performance of models.

8.1 Data quality

The quality of the data used in fault prediction has significant potential to undermine the efficacy of a model. Data quality is complex and many aspects of data are important to ensuring reliable predictions. Unfortunately, it is often difficult to assess the quality of data used in studies especially as many studies report very little about the data they use. Without good quality data, clearly reported, it is difficult to have confidence in the predictive results of studies.

The results of our assessment show that data quality is an issue in many studies. In fact many studies failed our synthesis assessment on the basis that they either reported insufficient information about the context of their data or about the collection of that data. Many studies explicitly acknowledge the importance of data quality (e.g. Jiang et al [[64]]).

Collecting good quality data is very hard. This is partly reflected by the number of studies which failed our assessment by not adequately explaining how they had collected their independent or dependent data. Fault data collection has been previously shown to be particularly hard to collect, usually because fault data is either not directly recorded or recorded poorly [14]. Collecting data is made more challenging because large data sets are usually necessary for reliable fault prediction. Jiang et al [[64]] investigate the impact that the size of the training and test data set has on the accuracy of predictions. Tosun et al [[176]] presents a useful insight into the real challenges associated with every aspect of fault prediction, but particularly on the difficulties of collecting reliable metrics and fault data. Once collected data is usually noisy and often needs to be cleaned (e.g. outliers and missing values dealt with). Very few studies report any data cleaning (even in our 35 finally included studies).

The balance of data (i.e. the number of faulty as opposed to non-faulty units) on which models are trained and tested is acknowledged by a few studies as fundamental to the reliability of models (see Appendix F for more information on data imbalance). Indeed, across the 18 studies we analysed in detail, some of those performing best are based on data with a good proportion of faulty units (e.g. [[164]], [[37]], [[11]], [[74]]). Our analysis also suggests that data imbalance in relation to specific modeling techniques (e.g. C4.5) may be related to poor performance (e.g. [[8]] and [[9]]). Several studies specifically investigated the impact of data balance and propose techniques to deal with it. For example, Khoshgoftaar et al [[76]] and Shivaji et al [[164]] present techniques for ensuring reliable data distributions. Schroter et al [[154]] base their training set on the faultiest parts of the system. Similarly, Seiffert et al [[156]] present data sampling and boosting techniques to address data imbalance. Data imbalance is explored further in Fioravanti and

Nesi [[13]] and Zhang et al [[200]]. Many studies seem to lack awareness of the need to account for data imbalance. As a consequence the impact of imbalanced data on the real performance of models can be hidden by the performance measures selected. This is especially true where the balance of data is not even reported. Readers are then not able to account for the degree of imbalanced data in their interpretation of predictive performance.

8.2 Measuring the predictive performance of models

There are many ways in which the performance of a prediction model can be measured. Indeed, many different categorical and continuous performance measures are used in our 35 studies. There is no one best way to measure the performance of a model. This depends on: the class distribution of the training data, how the model has been built and how the model will be used. For example, the importance of measuring misclassification will vary depending on, for example, the application.

Performance comparison across studies is only possible if studies report a set of uniform measures. Furthermore any uniform set of measures should give a full picture of correct and incorrect classification. To make models reporting categorical results most useful, we believe that the raw confusion matrix on which their performance measures are derived should be reported. This confusion matrix data would allow other researchers and potential users to calculate the majority of other measures. Pizzi et al [[133]] provide a very usable format for presenting a confusion matrix. Some studies present many models and it is not practical to report the confusion matrices for all these. Menzies et al [[114]] suggest a useful way in which data from multiple confusion matrices may be effectively reported. Alternatively, Lessmann [[97]] recommends that ROC curves and AUC are most useful when comparing the ability of modeling techniques to cope with different datasets. Either of these approaches adopted widely would make studies more useful in the future. Comparing across studies reporting continuous results is currently even more difficult and is the reason we were unable to present comparative box plots across these studies. To enable cross comparison we recommend that continuous studies report Average Relative Error (ARE) in addition to any preferred measures presented.

The impact of performance measurement has been picked up in many studies. Zhou et al [[203]] report that the use of some measures, in the context of a particular model, can present a misleading picture of predictive performance and undermine the reliability of predictions. Arisholm et al [[9]] discuss how model performance varies depending on how it is measured. There is an increasing focus on identifying effective ways to measure the performance of models. Cost and/or effort aware measurement is now a significant strand of interest in prediction measurement. This takes into account the cost/effort of falsely identifying modules and has been increasingly reported as useful. The concept of cost-effectiveness measurement originated with the Simula group (e.g. Arisholm et al [[9]]), but has more recently been taken up and developed by other researchers, for example Nagappan et al [[120]] and Mende and Koschke [[109]].

8.3 Fault severity

Few studies incorporate fault severity into their measurement of predictive performance. Although some faults are more important to identify than others, few models differentiate between the faults predicted. In fact Shatnawi and Li's [[160]] was the only study in the final 35 to use fault severity in their model. They report a model able to predict high and medium severity faults (these levels of severity are based on those reported in Bugzilla by Eclipse developers). Lamkanfi et al [16], Singh et al [[167]] and Zhou and Leung [[202]] are other studies which have also investigated severity. This lack of studies that consider severity is probably because, although acknowledged to be important, severity is considered a difficult concept to measure. For example, Menzies et al [244] say that severity is too vague to reliably investigate, Nikora [[126]]says that "without a widely agreed definition of severity we cannot reason about it" and Ostrand et al [17] state that severity levels are highly subjective and can be inaccurate and inconsistent. These problems of how to measure and collect reliable severity data may limit the usefulness of fault prediction models. Companies developing non-critical systems may want to prioritise their fault finding effort only on the most severe faults.

8.4 The reporting of fault prediction studies

Our results suggest that overall fault prediction studies are reported poorly. Out of the 208 studies initially included in our review, only 35 passed our assessment criteria. Many of these criteria are focused on checking that studies report basic details about the study. Without a basic level of information reported it is hard to have confidence in a study. Our results suggest that many studies are failing to report information which is considered essential when reporting empirical studies in other domains. The poor reporting of studies has consequences for both future researchers and potential users of models: it is difficult for researchers to meta-analyse across studies and it is difficult to replicate studies; it is also difficult for users to identify suitable models for implementation.

8.5 NASA data

NASA's publicly available software metrics data have proved very popular in developing fault prediction models. We identify all 62 studies which use NASA data in the reference list of the 208 included studies. The NASA data is valuable as it enables studies using different modelling techniques and independent variables to be compared to others using the same data set. It also allows studies to be replicated. A meta-analysis of the studies using NASA data would be valuable. However, although the repository holds many metrics and is publicly available it does have limitations. It is not possible to explore the source code and the contextual data is not comprehensive (e.g. no data on maturity is available). It is also not always possible to identify if any changes have been made to the extraction and computation mechanisms over time. In addition the data may suffer from noise [[83]] and other important anomalies [18]. It is also questionable whether a model that works well on the NASA data will work on a different type of system; as Menzies et al. [[112]] point out, NASA works in a unique niche market developing software which is not typical of the generality of software systems. However Turhan et al [[181]] have demonstrated that models built on NASA data are useful for predicting faults in software embedded in white goods.

9 Conclusions

Fault prediction is an important topic in software engineering. Fault prediction models have the potential to improve the quality of systems and reduce the costs associated with delivering those systems. As a result of this many fault prediction studies in software engineering have been published. Our analysis of 208 of these studies shows that the vast majority are less useful than they could be. Most studies report insufficient contextual and methodological information to enable full understanding of a model. This makes it difficult for potential model users to select a model to match their context and few models have transferred into industrial practice. It also makes it difficult for other researchers to meta-analyse across models to identify the influences on predictive performance. Overall a great deal of effort has gone into models that are of limited use to either practitioners or researchers.

The set of criteria we present identify a set of essential contextual and methodological details that fault prediction studies should report. These go some way towards addressing the need identified by Myrtveit et al [20] for "more reliable research procedures before we can have confidence in the conclusions of comparative studies". Our criteria should be used by future model builders. They should also be used by Journal and Conference reviewers. This would ensure that future studies are built reliably, and reported comparably with other such reliable studies. Of the 208 studies we reviewed, only 35 satisfied our criteria and reported essential contextual and methodological details.

We analysed these 35 studies to determine what impacts on model performance in terms of the context of models, the independent variables used by models and the modelling techniques on which they were built. Our results suggest that models which perform well tend to be built in a context where the systems are larger. We found no evidence that the maturity of systems or the language used is related to predictive performance. But we did find some evidence to suggest that some application domains (e.g. embedded systems) may be more difficult to build reliable prediction models for. The independent variables used by models performing well seem to be sets of metrics (e.g. combinations of process, product and people-based metrics). We found evidence that where models use KLOC as their independent variable, they perform no worse than where only single sets of other static code metrics are used. In addition models which perform well tend to use simple, easy to use modelling techniques like Naïve Bayes or Logistic Regression. More complex modelling techniques, such as Support Vector Machines, tend to be used by models which perform relatively less well.

The methodology used to build models seems to be influential to predictive performance. The models which performed well seemed to optimise three aspects of the model. First, the choice of data was optimised. In particular, successful models tend to be trained on large data sets which contain a relatively high proportion of faulty units. Second, the choice of independent variables was optimised. A large range of metrics were used on which feature selection was applied. Third, the modelling technique was optimised. The default parameters were adjusted to ensure that the technique would perform effectively on the data provided.

Overall we conclude that many good fault prediction studies have been reported in software engineering (e.g. the 35 which passed our assessment criteria). Some of these studies are of exceptional quality, for example Shivaji et al [[164]]. However there remain many open questions about how to build effective fault prediction models for software systems. We need more studies which are based on a reliable methodology and which consistently report the context in which models are built and the methodology used to build them. A larger set of such studies will enable reliable cross study meta-analysis of model performance. It will also give practitioners the confidence to appropriately select and apply models to their systems. Without this increase in reliable models that are appropriately reported, fault prediction will continue to have limited impact on the quality and cost of industrial software systems.

Acknowledgements

We are grateful to the UK's Engineering and Physical Science Research Council who supported this research at Brunel University under grant EPSRC EP/E063039/1 and to Science Foundation Ireland grant 3/CE2/I303_1 who partially supported this work at Lero. We are also grateful to Dr Sue Black and Dr Paul Wernick who provided input to the early stages of the work reported in this paper and to Professor Martin Shepperd for his suggestions throughout the work. We are also grateful for the detailed and insightful comments from the reviewers that enabled us to significantly improve the quality of this paper.

Main References

[1] N. Fenton and M. Neil, "A critique of software defect prediction models," *Software Engineering, IEEE Trans on*, vol. 25, no. 5, pp. 675–689, 1999.

[2] C. Catal and B. Diri, "A systematic review of software fault prediction studies," *Expert Systems with Applications*, vol. 36, no. 4, pp. 7346–7354, 2009.

[3] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering (version 2.3)," *Keele University, UK, Tech. Rep. EBSE Technical Report EBSE-2007-01*, 2007.

[4] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *Software Engineering, IEEE Transactions on*, vol. 33, no. 1, pp. 33–53, 2007.

[5] B. Kitchenham, "What's up with software metrics? -a preliminary mapping study," *Journal of Systems and Software*, vol. 83, no. 1, pp. 37–51, 2010.

[6] K. Petersen and C. Wohlin, "Context in industrial software engineering research," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE Computer Society, 2009, pp. 401–404.

[7] D. Bowes and T. Hall, "SLuRp: A web enabled database for effective management of systematic literature reviews," University of Hertfordshire, Tech. Rep. 510, 2011.

[8] D. S. Cruzes and T. Dybå, "Research synthesis in software engineering: A tertiary study," *Inf. Softw. Technol.*, vol. 53, pp. 440–455, May 2011.

[9] R. Rosenthal and M. DiMatteo, "Meta-analysis: Recent developments in quantitative methods for literature reviews," *Annual review of psychology*, vol. 52, no. 1, pp. 59–82, 2001.

[10] B. Turhan, T. Menzies, A. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009.

[11] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intell. Data Anal*, vol. 6, no. 5, pp. 429–449, 2002.

[12] C. Hsu, C. Chang, and C. Lin, "A practical guide to support vector classification," *Department of Computer Science and Information Engineering, National Taiwai University*, 2003.

[13] N. Cristianini and J. Shawe-Taylor, *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge university press, 2006.

[14] T. Hall, D. Bowes, G. Liebchen, and P. Wernick, "Evaluating three approaches to extracting fault data from software change repositories," in *International Conference on Product Focused Software Development and Process Improvement (PROFES)*. Springer, 2010, pp. 107–115.

[15] W. Liu, S. Chawla, D. A. Cieslak, and N. V. Chawla, "A robust decision tree algorithm for imbalanced data sets," in *SDM*. SIAM, 2010, pp. 766–777.

[16] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, 2010, pp. 1–10.

[17] T. Ostrand, E. Weyuker, and R. Bell, "Where the bugs are," in *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 4. ACM, 2004, pp. 86–96.

[18] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Further thoughts on precision," in *Evaluation and Assessment in Software Engineering (EASE)*, 2011.

[19] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors'"," *Software Engineering, IEEE Transactions on*, vol. 33, no. 9, pp. 637 –640, sept. 2007.

[20] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Transactions on Software Engineering*, pp. 380–391, 2005.

[21] G. Liebchen and M. Shepperd, "Data sets and data quality in software engineering," in *Proceedings of the 4th international workshop on Predictor models in software engineering*. ACM, 2008, pp. 39–44.

[22] H. Zhang and X. Zhang, "Comments on "data mining static code attributes to learn defect predictors"," *Software Engineering, IEEE Transactions on*, vol. 33, no. 9, pp. 635 –637, sept. 2007.

[23] G. Batista, R. Prati, and M. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[24] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K. Matsumoto, "The effects of over and under sampling on fault-prone module detection," in *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, sept. 2007, pp. 196 – 204.

[25] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Software defect prediction using static code metrics underestimates defect-proneness," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE, pp. 1–7.

[26] T. Ostrand and E. Weyuker, "How to measure success of fault prediction models," in *Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting*. ACM, 2007, pp. 25–30.

[27] D. Bowes and D. Gray, "Recomputing the confusion matrix for prediction studies reporting categorical output," University of Hertfordshire, Tech. Rep. 509, 2011.

[28] N. V. Chawla, N. Japkowicz, and A. Kotcz, "Editorial: special issue on learning from imbalanced data sets," *SIGKDD Explorations*, vol. 6, no. 1, pp. 1–6, 2004.

[29] S. Visa and A. Ralescu, "Fuzzy classifiers for imbalanced, complex classes of varying size," in *Information Processing and Management of Uncertainty in Knowledge-Based Systems*. 2004, pp. 393–400.

[30] H. He and E. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1263–1284, 2008.

References for the 208 included SLR papers [[1-208]]

References from this list are cited using the format [[ref#]]). Each reference is followed by a code indicating the status of the paper in terms of whether it passed (P) or failed (F) our assessment. An indication is also given as to the assessment phase a paper failed (1, 2, 3, 4 or 5). The use of NASA data by studies is also indicated (N). A paper (n) failing an assessment criterion in phase 2 which used NASA data would be coded: (Paper=n; Status=F, Phase=2, Data=N)

Please report possible problems with our assessment of these papers to: tracy.hall@brunel.ac.uk

[[1]] R. Abreu and R. Premraj, "How developer communication frequency relates to bug introducing changes," in *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*. ACM, 2009, pp. 153–158. (Paper=1, Status=F, Phase=1)

[[2]] W. Afzal, "Using faults-slip-through metric as a predictor of fault-proneness," in *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, 2010, pp. 414–422. (Paper=2, Status=F, Phase=2)

[[3]] W. Afzal and R. Torkar, "A comparative evaluation of using genetic programming for predicting fault count data," in *Software Engineering Advances, 2008. ICSEA '08. The Third International Conference on*, 2008, pp. 407–414. (Paper=3, Status=F, Phase=2)

[[4]] W. Afzal, R. Torkar, and R. Feldt, "prediction of fault count data using genetic programming," in *Multitopic Conference, 2008. INMIC 2008. IEEE International*, 2008, pp. 349–356. (Paper=4, Status=F, Phase=2)

[[5]] S. Amasaki, Y. Takagi, O. Mizuno, and T. Kikuno, "A bayesian belief network for assessing the likelihood of fault content," in *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on*, nov. 2003, pp. 215 – 226. (Paper=5, Status=F, Phase=2)

[[6]] C. Andersson and P. Runeson, "A replicated quantitative analysis of fault distributions in complex software systems," *Software Engineering, IEEE Transactions on*, vol. 33, no. 5, pp. 273 –286, may 2007. (Paper=6, Status=F, Phase=1)

[[7]] E. Arisholm and L. Briand, "Predicting fault-prone components in a java legacy system," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 2006, pp. 8–17. (Paper=7, Status=F, Phase=4)

[[8]] E. Arisholm, L. C. Briand, and M. Fuglerud, "Data mining techniques for building fault-proneness models in telecom java software," in *Software Reliability, 2007. ISSRE '07. The 18th IEEE International Symposium on*, nov. 2007, pp. 215 –224. (Paper=8, Status=P)

[[9]] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010. (Paper=9, Status=P)

[[10]] N. Ayewah, W. Pugh, J. Morgenthaler, J. Penix, and Y. Zhou, "Evaluating static analysis defect warnings on production software," in *Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. ACM, 2007, pp. 1–8. (Paper=10, Status=F, Phase=1)

[[11]] R. Bell, T. Ostrand, and E. Weyuker, "Looking for bugs in all the right places," in *Proceedings of the 2006 international symposium on Software testing and analysis*. ACM, 2006, pp. 61–72. (Paper=11, Status=P)

[[12]] P. Bellini, I. Bruno, P. Nesi, and D. Rogai, "Comparing fault-proneness estimation models," in *Engineering of Complex Computer Systems, 2005. ICECCS 2005. Proceedings. 10th IEEE International Conference on*, june 2005, pp. 205 – 214. (Paper=12, Status=F, Phase=2)

[[13]] A. Bernstein, J. Ekanayake, and M. Pinzger, "Improving defect prediction using temporal features and non linear models," in *Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting*, ser. IWPSE '07. New York, NY, USA: ACM, 2007, pp. 11–18. (Paper=13, Status=F, Phase=2)

[[14]] M. Bezerra, A. Oliveira, and S. Meira, "A constructive rbf neural network for estimating the probability of defects in software modules," in *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, aug. 2007, pp. 2869 –2874. (Paper=14, Status=F, Phase=2, Data=N)

[[15]] S. Bibi, G. Tsoumakas, I. Stamelos, and I. Vlahvas, "Software defect prediction using regression via classification," in *Computer Systems and Applications, 2006. IEEE International Conference on.*, 8, 2006, pp. 330 – 336. (Paper=15, Status=P)

[[16]] D. Binkley, H. Feild, D. Lawrie, and M. Pighin, "Software fault prediction using language processing," in *Testing: Academic and Industrial Conference Practice and Research Techniques. TAICPART-MUTATION 2007*, sept. 2007, pp. 99 –110. (Paper=16, Status=F, Phase=1)

[[17]] D. Binkley, H. Feild, D. Lawrie, and M. Pighin, "Increasing diversity: Natural language measures for software fault prediction," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1793–1803, 2009. (Paper=17, Status=F, Phase=1)

[[18]] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu, "Putting it all together: Using socio-technical networks to predict failures," in *20th International Symposium on Software Reliability Engineering*. IEEE, 2009, pp. 109–119. (Paper=18, Status=P)

[[19]] G. Boetticher, *Advanced machine learner applications in software engineering*. Hershey, PA, USA: Idea Group Publishing, 2006, ch. Improving credibility of machine learner models in software engineering, pp. 52 – 72. (Paper=19, Status=F, Phase=2, Data=N)

[[20]] L. Briand, W. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," *Software Engineering, IEEE Transactions on*, vol. 28, no. 7, pp. 706 – 720, jul 2002. (Paper=20, Status=F, Phase=2)

[[21]] B. Caglayan, A. Bener, and S. Koch, "Merits of using repository metrics in defect prediction for open source projects," in *Emerging Trends in Free/Libre/Open Source Software Research and Development, 2009. FLOSS '09. ICSE Workshop on*, 2009, pp. 31–36. (Paper=21, Status=P)

[[22]] G. Calikli, A. Tosun, A. Bener, and M. Celik, "The effect of granularity level on software defect prediction," in *Computer and Information Sciences, 2009. ISCIS 2009. 24th International Symposium on*, sept. 2009, pp. 531 –536. (Paper=22, Status=F, Phase=2)

[[23]] C. Catal, B. Diri, and B. Ozumut, "An artificial immune system approach for fault prediction in object-oriented software," in *Dependability of Computer Systems. DepCoS-RELCOMEX '07. 2nd International Conference on*, 2007, pp.238–245. (Paper=23, Status=F, Phase=2, Data=N)

[[24]] E. Ceylan, F. Kutlubay, and A. Bener, "Software defect identification using machine learning techniques," in *Software Engineering and Advanced Applications, 2006. SEAA '06. 32nd EUROMICRO Conference on*, 2006, pp. 240 –247. (Paper=24, Status=F, Phase=2)

[[25]] V. U. Challagulla, F. B. Bastani, and I.-L. Yen, "A unified framework for defect data analysis using the mbr technique," in *Tools with Artificial Intelligence,. ICTAI '06. 18th IEEE International Conference on*, nov. 2006, pp. 39 –46. (Paper=25, Status=F, Phase=2, Data=N)

[[26]] V. Challagulla, F. Bastani, I.-L. Yen, and R. Paul, "Empirical assessment of machine learning based software defect prediction techniques," in *Object-Oriented Real-Time Dependable Systems, 10th IEEE Intern'l Workshop on*. 2005, pp263–270. (Paper=26, Status=F, Phase=2, Data=N)

[[27]] J. Cong, D. En-Mei, and Q. Li-Na, "Software fault prediction model based on adaptive dynamical and median particle swarm optimization," in *Multimedia and Information Technology (MMIT), Second Inter'l Conf on*, vol. 1, 2010, pp. 44–47. (Paper=27, Status=F, Phase=2, Data=N)

[[28]] C. Cruz and A. Erika, "Exploratory study of a uml metric for fault prediction," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM, 2010, pp. 361–364. (Paper=28, Status=F, Phase=4)

[[29]] C. Cruz, A. Erika, and O. Koichiro, "Towards logistic regression models for predicting fault-prone code across software projects," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 460–463. (Paper=29, Status=P)

[[30]] V. Dallmeier and T. Zimmermann, "Extraction of bug localization benchmarks from history," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ASE '07., NY, USA: ACM, 2007, pp. 433–436. (Paper=30, Status=F, Phase=1)

[[31]] M. D'Ambros, M. Lanza, and R. Robbes, "On the relationship between change coupling and software defects," in *Reverse Engineering, 2009. WCRE '09. 16th Working Conference on*, oct. 2009, pp. 135 –144. (Paper=31, Status=P)

[[32]] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, 2010, pp. 31–41. (Paper=32, Status=P)

[[33]] A. B. de Carvalho, A. Pozo, S. Vergilio, and A. Lenz, "Predicting fault proneness of classes trough a multiobjective particle swarm optimization algorithm," in *Tools with Artificial Intelligence, 2008. ICTAI '08. 20th IEEE International Conference on*, vol. 2, 2008, pp. 387–394. (Paper=33, Status=F, Phase=2, Data=N)

[[34]] A. B. de Carvalho, A. Pozo, and S. R. Vergilio, "A symbolic fault-prediction model based on multiobjective particle swarm optimization," *Journal of Systems and Software*, vol. 83, no. 5, pp. 868–882, 2010. (Paper=34, Status=F, Phase=2, Data=N)

[[35]] G. Denaro, "Estimating software fault-proneness for tuning testing activities," in *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, 2000, pp. 704 –706. (Paper=35, Status=F, Phase=2)

[[36]] G. Denaro, S. Morasca, and M. Pezzè, "Deriving models of software fault-proneness," in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, ser. SEKE '02. New York, NY, USA: ACM, 2002, pp. 361–368. (Paper=36, Status=F, Phase=2)

[[37]] G. Denaro and M. Pezzè, "An empirical evaluation of fault-proneness models," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. New York, NY, USA: ACM, 2002, pp. 241–251. (Paper=37, Status=P)

[[38]] Z. Dianqin and W. Zhongyuan, "The application of gray-prediction theory in the software defects management," in *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on*, 2009, pp. 1–5. (Paper=38, Status=F, Phase=2)

[[39]] K. El Emam, W. Melo, and J. Machado, "The prediction of faulty classes using object-oriented design metrics," *Journal of Systems and Software*, vol. 56, no. 1, pp. 63–75, 2001. (Paper=39, Status=F, Phase=2)

[[40]] K. Elish and M. Elish, "Predicting defect-prone software modules using support vector machines," *Journal of Systems and Software*, vol. 81, no. 5, pp. 649–660, 2008. (Paper=40, Status=F, Phase=2, Data=N)

[[41]] N. Fenton, M. Neil, W. Marsh, P. Hearty, L. Radlinski, and P. Krause, "Project data incorporating qualitative factors for improved software defect prediction," in *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, may 2007, p. 2. (Paper=41, Status=F, Phase=2)

[[42]] N. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *Software Engineering, IEEE Transactions on*, vol. 26, no. 8, pp. 797 –814, aug 2000. (Paper=42, Status=F, Phase=1)

[[43]] F. Fioravanti and P. Nesi, "A study on fault-proneness detection of object-oriented systems," in *Software Maintenance and Reengineering, 2001. Fifth European Conference on*, 2001, pp. 121 –130. (Paper=43, Status=F, Phase=2)

[[44]] K. Gao and T. Khoshgoftaar, "A comprehensive empirical study of count models for software fault prediction," *Reliability, IEEE Transactions on*, vol. 56, no. 2, pp. 223 –236, june 2007. (Paper=44, Status=F, Phase=2)

[[45]] N. Gayatri, S. Nickolas, A. V. Reddy, and R. Chitra, "Performance analysis of datamining algorithms for software quality prediction," in *Advances in Recent Technologies in Communication and Computing, 2009. ARTCom '09. International Conference on*. accept, 2009, pp. 393–395. (Paper=45, Status=F, Phase=2, Data=N)

[[46]] I. Gondra, "Applying machine learning to software fault-proneness prediction," *Journal of Systems and Software*, vol. 81, no. 2, pp. 186–195, 2008. (Paper=46, Status=F, Phase=1)

[[47]] T. Graves, A. Karr, J. Marron, and H. Siy, "Predicting fault incidence using software change history," *Software Engineering, IEEE Transactions on*, vol. 26, no. 7, pp. 653 –661, jul 2000. (Paper=47, Status=F, Phase=1)

[[48]] D. Gray, D. Bowes, N. Davey, S. Yi, and B. Christianson, "Software defect prediction using static code metrics underestimates defect-proneness," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*, 2010, pp. 1–7. (Paper=48, Status=F, Phase=1)

[[49]] L. Guo, B. Cukic, and H. Singh, "Predicting fault prone modules by the dempster-shafer belief networks," in *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, oct. 2003, pp. 249 – 252. (Paper=49, Status=F, Phase=2, Data=N)

[[50]] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of fault-proneness by random forests," in *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on*, nov. 2004, pp. 417 – 428. (Paper=50, Status=F, Phase=2, Data=N)

[[51]] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *Software Engineering, IEEE Transactions on*, vol. 31, no. 10, pp. 897 – 910, oct. 2005. (Paper=51, Status=P)

[[52]] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, 2009, pp. 78–88. (Paper=52, Status=F, Phase=6)

[[53]] A. Hassan and R. Holt, "The top ten list: dynamic fault prediction," in *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, sept. 2005, pp. 263 – 272. (Paper=53, Status=F, Phase=1)

[[54]] Y. Higo, K. Murao, S. Kusumoto, and K. Inoue, "Predicting fault-prone modules based on metrics transitions," in *Proceedings of the 2008 workshop on Defects in large software systems*. ACM, 2008, pp. 6–10. (Paper=54, Status=F, Phase=1)

[[55]] T. Holschuh, M. Pauser, K. Herzig, T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects in sap java code: An experience report," in *Software Engineering - Companion Volume, 2009. ICSE-Companion. 31st Intern'l Conf on*, 2009, pp.172–181. (Paper=55, Status=F, Phase=2)

[[56]] Z. Hongyu, "An investigation of the relationships between lines of code and defects," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, 2009, pp. 274–283. (Paper=56, Status=P, Data=N)

[[57]] D. Hovemeyer and W. Pugh, "Finding bugs is easy," in *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, ser. OOPSLA '04, NY, USA: ACM, 2004, pp. 132–136. (Paper=57, Status=F, Phase=3)

[[58]] L. Hribar and D. Duka, "Software component quality prediction using knn and fuzzy logic," in *MIPRO, 2010 Proceedings of the 33rd International Convention*, 2010, pp. 402–408. (Paper=58, Status=F, Phase=1)

[[59]] W. Huanjing, T. M. Khoshgoftaar, and A. Napolitano, "A comparative study of ensemble feature selection techniques for software defect prediction," in *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, 2010, pp. 135–140. (Paper=59, Status=F, Phase=4, Data=N)

[[60]] L. Jiang, Z. Su, and E. Chiu, "Context-based detection of clone-related bugs," 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*, ser. ESEC-FSE '07. New York, NY, USA: ACM, 2007, pp. 55–64. (Paper=60, Status=F, Phase=1)

[[61]] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Software Engineering*, vol. 13, no. 5, pp. 561–595, 2008. (Paper=61, Status=F, Phase=2, Data=N)

[[62]] Y. Jiang, B. Cukic, and T. Menzies, "Fault prediction using early lifecycle data," in *Software Reliability, 2007. ISSRE '07. The 18th IEEE International Symposium on*, nov. 2007, pp. 237 –246. (Paper=62, Status=F, Phase=2, Data=N)

[[63]] Y. Jiang, B. Cukic, and T. Menzies, "Cost curve evaluation of fault prediction models," in *19th International Symposium on Software Reliability Engineering (ISSRE 2008)*. 11-14 November 2008, Seattle/Redmond, WA,USA: IEEE Computer Society, 2008, pp. 197–206. (Paper=63, Status=F, Phase=2, Data=N)

[[64]] Y. Jiang, J. Lin, B. Cukic, and T. Menzies, "Variance analysis in software fault prediction models," in *SSRE 2009, 20th International Symposium on Software Reliability Engineering*. Mysuru, Karnataka, India, 16-19 November 2009: IEEE Computer Society, 2009, pp. 99–108. (Paper=64, Status=F, Phase=2, Data=N)

[[65]] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ser. ASE '05. New York, NY, USA: ACM, 2005, pp. 273–282. (Paper=65, Status=F, Phase=1)

[[66]] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. , NY, USA: ACM, 2002, pp. 467–477. (Paper=66, Status=F, Phase=1)

[[67]] H. Joshi, C. Zhang, S. Ramaswamy, and C. Bayrak, "Local and global recency weighting approach to bug prediction," in *Mining Software Repositories, 2007. ICSE Workshops MSR '07. Fourth International Workshop on*, may 2007, p. 33. (Paper=67, Status=F, Phase=2)

[[68]] L. Jun, X. Zheng, Q. Jianzhong, and L. Shukuan, "A defect prediction model for software based on service oriented architecture using expert cocomo," in *Control and Decision Conference, 2009. CCDC '09. Chinese*. accept, 2009, pp. 2591–2594. (Paper=68, Status=F, Phase=1)

[[69]] Y. Kamei, S. Matsumoto, A. Monden, K. i. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting common bug prediction findings using effort-aware models," in *Software Maintenance (ICSM), IEEE International Conference on*. accept, 2010, pp. 1–10. (Paper=69, Status=P)

[[70]] K. Kaminsky and G. Boetticher, "Building a genetically engineerable evolvable program (geep) using breadth-based explicit knowledge for predicting software defects," in *Fuzzy Information, 2004. Processing NAFIPS '04. IEEE Annual Meeting of the*, vol. 1, june 2004, pp. 10 – 15 Vol.1. (Paper=70, Status=F, Phase=1)

[[71]] S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, "Object oriented software quality prediction using general regression neural networks," *SIGSOFT Softw. Eng. Notes*, vol. 29, pp. 1–6, September 2004. (Paper=71, Status=F, Phase=2)

[[72]] S. Kanmani, V. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, "Object-oriented software fault prediction using neural networks," *Information and Software Technology*, vol. 49, no. 5, pp. 483–492, 2007. (Paper=72, Status=F, Phase=2)

[[73]] Y. Kastro and A. Bener, "A defect prediction method for software versioning," *Software Quality Journal*, vol. 16, no. 4, pp. 543–562, 2008. (Paper=73, Status=P)

[[74]] A. Kaur and R. Malhotra, "Application of random forest in predicting fault-prone classes," in *Advanced Computer Theory and Engineering, 2008. ICACTE '08. International Conference on*, 2008, pp. 37–43. (Paper=74, Status=P)

[[75]] A. Kaur, P. S. Sandhu, and A. S. Bra, "Early software fault prediction using real time defect data," in *Machine Vision, 2009. ICMV '09. Second International Conference on*. accept, 2009, pp. 242–245. (Paper=75, Status=F, Phase=2, Data=N)

[[76]] T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute selection and imbalanced data: Problems in software defect prediction," in *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, vol. 1. accept, 2010, pp. 137–144. (Paper=76, Status=P)

[[77]] T. Khoshgoftaar, E. Allen, and J. Busboom, "Modeling software quality: the software measurement analysis and reliability toolkit," in *Tools with Artificial Intelligence, 2000. ICTAI 2000. Proceedings. 12th IEEE International Conf on*, 2000, pp.54–61. (Paper=77, Status=F, Phase=2)

[[78]] T. Khoshgoftaar, K. Gao, and R. Szabo, "An application of zero-inflated poisson regression for software fault prediction," in *Software Reliability Engineering, 2001. ISSRE 2001. Procs. 12th International Symposium on*, nov. 2001, pp. 66 – 73. (Paper=78, Status=F, Phase=2)

[[79]] T. Khoshgoftaar, E. Geleyn, and K. Gao, "An empirical study of the impact of count models predictions on module-order models," in *Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on*, 2002, pp. 161 – 172. (Paper=79, Status=F, Phase=2)

[[80]] T. Khoshgoftaar and N. Seliya, "Improving usefulness of software quality classification models based on boolean discriminant functions," in *Software Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th International Symposium on*, 2002, pp. 221 – 230. (Paper=80, Status=F, Phase=2)

[[81]] T. Khoshgoftaar and N. Seliya, "Software quality classification modeling using the sprint decision tree algorithm," in *Tools with Artificial Intelligence, 2002. (ICTAI 2002). Proceedings. 14th IEEE International Conference on*, 2002, pp. 365 – 374. (Paper=81, Status=F, Phase=2)

[[82]] T. Khoshgoftaar and N. Seliya, "Tree-based software quality estimation models for fault prediction," in *Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on*, 2002, pp. 203 – 214. (Paper=82, Status=F, Phase=2)

[[83]] T. Khoshgoftaar and N. Seliya, "Comparative assessment of software quality classification techniques: An empirical case study," *Empirical Software Engineering*, vol. 9, no. 3, pp. 229–257, 2004. (Paper=83, Status=P)

[[84]] T. Khoshgoftaar, N. Seliya, and K. Gao, "Assessment of a new three-group software quality classification technique: An empirical case study," *Empirical Software Engineering*, vol. 10, no. 2, pp. 183–218, 2005. (Paper=84, Status=F, Phase=2)

[[85]] T. Khoshgoftaar, V. Thaker, and E. Allen, "Modeling fault-prone modules of subsystems," in *Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium on*, 2000, pp. 259 –267. (Paper=85, Status=F, Phase=2)

[[86]] T. Khoshgoftaar, X. Yuan, E. Allen, W. Jones, and J. Hudepohl, "Uncertain classification of fault-prone software modules," *Empirical Software Engineering*, vol. 7, no. 4, pp. 297–318, 2002. (Paper=86, Status=P)

[[87]] S. Kim, K. Pan, and E. Whitehead Jr, "Memories of bug fixes," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2006, pp. 35–45. (Paper=87, Status=F, Phase=2)

[[88]] S. Kim, T. Zimmermann, E. Whitehead Jr, and A. Zeller, "Predicting faults from cached history," in *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 2007, pp. 489–498. (Paper=88, Status=F, Phase=4)

[[89]] S. Kim, T. Zimmermann, K. Pan, and E. Whitehead, "Automatic identification of bug-introducing changes," in *Automated Software Engineering, 2006. ASE '06. 21st IEEE/ACM International Conference on*, sept. 2006, pp. 81 –90. (Paper=89, Status=F, Phase=1)

[[90]] M. Kläs, F. Elberzhager, J. Münch, K. Hartjes, and O. von Graevemeyer, "Transparent combination of expert and measurement data for defect prediction: an industrial case study," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM, 2010, pp. 119–128. (Paper=90, Status=F, Phase=2)

[[91]] M. Kläs, H. Nakao, F. Elberzhager, and J. Münch, "Predicting defect content and quality assurance effectiveness by combining expert judgment and defect data - a case study," in *Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on*, 2008, pp. 17–26. (Paper=91, Status=F, Phase=2)

[[92]] P. Knab, M. Pinzger, and A. Bernstein, "Predicting defect densities in source code files with decision tree learners," in *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 2006, pp. 119–125. (Paper=92, Status=P)

[[93]] A. Koru and H. Liu, "Building effective defect-prediction models in practice," *Software, IEEE*, vol. 22, no. 6, pp. 23 – 29, nov.-dec. 2005. (Paper=93, Status=F, Phase=2, Data=N)

[[94]] A. Koru, D. Zhang, and H. Liu, "Modeling the effect of size on defect proneness for open-source software," in *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, may 2007, p. 10. (Paper=94, Status=F, Phase=1)

[[95]] O. Kutlubay, B. Turhan, and A. Bener, "A two-step model for defect density estimation," in *Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on*, aug. 2007, pp. 322 –332. (Paper=95, Status=F, Phase=2, Data=N)

[[96]] L. Layman, G. Kudrjavets, and N. Nagappan, "Iterative identification of fault-prone binaries using in-process metrics," in *Procs of the Second ACM-IEEE intern'l symp on Empirical software engineering and measurement*. ACM, 2008, pp. 206–212. (Paper=96, Status=F, Phase=4)

[[97]] S. Lessmannn, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *Software Engineering, IEEE Trans on*, vol. 34, no. 4, pp.485–496, july-aug. 2008. (Paper=97, Status=F, Phase=2, Data=N)

[[98]] P. L. Li, J. Herbsleb, M. Shaw, and B. Robinson, "Experiences and results from initiating field defect prediction and product test prioritization efforts at abb inc." in *Proceedings of the 28th international conference on Software engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 413–422. (Paper=98, Status=P)

[[99]] P. L. Li, M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam, "Empirical evaluation of defect projection models for widely-deployed production software systems," *SIGSOFT Softw. Eng. Notes*, vol. 29, pp. 263–272, October 2004. (Paper=99, Status=F, Phase=1)

[[100]] P. Li, J. Herbsleb, and M. Shaw, "Finding predictors of field defects for open source software systems in commonly available data sources: a case study of openbsd," in *Software Metrics, 11th IEEE International Symposium*, sept. 2005, pp. 10 pp. –32. (Paper=100, Status=F, Phase=1)

[[101]] Z. Li and M. Reformat, "A practical method for the software fault-prediction," in *Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on*, aug. 2007, pp. 659 –666. (Paper=101, Status=F, Phase=2, Data=N)

[[102]] Y. Ma, L. Guo, and B. Cukic, *Advances in Machine Learning Applications in Software Engineering*. IGI Global, 2006, ch. A statistical framework for the prediction of fault-proneness, pp. 237–265. (Paper=102, Status=F, Phase=2, Data=N)

[[103]] J. T. Madhavan and E. J. Whitehead, Jr., "Predicting buggy changes inside an integrated development environment," in *Procs of the 2007 OOPSLA workshop on eclipse technology eXchange*, ser. New York, NY, USA: ACM, 2007, pp. 36–40. (Paper=103, Status=F, Phase=4)

[[104]] A. Mahaweerawat, P. Sophatsathit, and C. Lursinsap, "Software fault prediction using fuzzy clustering and radial-basis function network," in *Proceedings of the International Conference on Intelligent Technologies, InTech/VJFuzzy*, 2002, pp. 304–313. (Paper=104, Status=F, Phase=2)

[[105]] A. Mahaweerawat, P. Sophatsathit, and C. Lursinsap, "Adaptive self-organizing map clustering for software fault prediction," in *Fourth international joint conference on computer science and software engineering, Thailand*, 2007, pp. 35–41. (Paper=105, Status=F, Phase=2)

[[106]] A. Mahaweerawat, P. Sophatsathit, C. Lursinsap, and P. Musilek, "Fault prediction in object-oriented software using neural network techniques," in *Proceedings of the InTech Conference, 2004*, 2004, pp. 2–4. (Paper=106, Status=F, Phase=2)

[[107]] A. Marcus, D. Poshyvanyk, and R. Ferenc, "Using the conceptual cohesion of classes for fault prediction in object-oriented systems," *Software Engineering, IEEE Transactions on*, vol. 34, no. 2, pp. 287 –300, march-april 2008. (Paper=107, Status=F, Phase=1)

[[108]] T. Mende and R. Koschke, "Revisiting the evaluation of defect prediction models," in *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. ACM, 2009, p. 7. (Paper=108, Status=F, Phase=2, Data=N)

[[109]] T. Mende and R. Koschke, "Effort-aware defect prediction models," in *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*, 2010, pp. 107–116. (Paper=109, Status=P, Data=N)

[[110]] T. Mende, R. Koschke, and M. Leszak, "Evaluating defect prediction models for a large evolving software system," in *Software Maintenance and Reengineering, 2009. CSMR '09. 13th European Conference on*, march 2009, pp. 247 –250. (Paper=110, Status=P)

[[111]] T. Menzies and J. Di Stefano, "How good is your blind spot sampling policy," in *High Assurance Systems Engineering, 2004. Proceedings. Eighth IEEE International Symposium on*, march 2004, pp. 129 – 138. (Paper=111, Status=F, Phase=2, Data=N)

[[112]] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *Software Engineering, IEEE Transactions on*, vol. 33, no. 1, pp. 2 –13, jan. 2007. (Paper=112, Status=F, Phase=2, Data=N)

[[113]] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. B. Bener, "Defect prediction from static code features: current results, limitations, new approaches," *Autom. Softw. Eng.*, vol. 17, no. 4, pp. 375–407, 2010. (Paper=113, Status=F, Phase=2, Data=N)

[[114]] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, and Y. Jiang, "Implications of ceiling effects in defect predictors," in *Proceedings of the 4th international workshop on Predictor models in software engineering*, ser. PROMISE '08. New York, NY, USA: ACM, 2008, pp. 47–54. (Paper=114, Status=F, Phase=2, Data=N)

[[115]] M. Mertik, M. Lenic, G. Stiglic, and P. Kokol, "Estimating software quality with advanced data mining techniques," in *Software Engineering Advances, International Conference on*, oct. 2006, p. 19. (Paper=115, Status=F, Phase=2, Data=N)

[[116]] O. Mizuno, S. Ikami, S. Nakaichi, and T. Kikuno, "Spam filter based approach for finding fault-prone software modules," in *Mining Software Repositories, 2007. ICSE Workshops MSR '07. Fourth International Workshop on*, may 2007, p. 4. (Paper=116, Status=P)

[[117]] O. Mizuno and T. Kikuno, "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*, ser. ESEC-FSE '07. New York, NY, USA: ACM, 2007, pp. 405–414. (Paper=117, Status=P)

[[118]] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Software Engineering ICSE. ACM/IEEE 30th International Conference on*. accept, 2008, pp. 181–190. (Paper=118, Status=P)

[[119]] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, may 2005, pp. 580 – 586. (Paper=119, Status=F, Phase=3)

[[120]] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy, "Change bursts as defect predictors," in *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*. accept, 2010, pp. 309–318. (Paper=120, Status=P)
[[121]] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th international conference on Software engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 284–292. (Paper=121, Status=F, Phase=2)
[[122]] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th international conference on Software engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 452–461. (Paper=122, Status=P)
[[123]] N. K. Nagwani and S. Verma, "Predictive data mining model for software bug estimation using average weighted similarity," in *Advance Computing Conference (IACC), 2010 IEEE 2nd International*. accept, 2010, pp. 373–378. (Paper=123, Status=F, Phase=1)
[[124]] A. Neufelder, "How to measure the impact of specific development practices on fielded defect density," in *Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium on*. IEEE, 2000, pp. 148–160. (Paper=124, Status=F, Phase=1)
[[125]] A. Nikora and J. Munson, "Developing fault predictors for evolving software systems," in *Software Metrics Symposium, 2003. Proceedings. Ninth International*, sept. 2003, pp. 338 – 350. (Paper=125, Status=F, Phase=1)
[[126]] A. Nikora and J. Munson, "The effects of fault counting methods on fault model quality," in *Computer Software and Applications Conference,. COMPSAC 2004. Proceedings of the 28th Annual International*, sept. 2004, pp. 192 –201 vol.1. (Paper=126, Status=F, Phase=1)
[[127]] A. Nugroho, M. R. V. Chaudron, and E. Arisholm, "Assessing uml design metrics for predicting fault-prone classes in a java system," in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. accept, 2010, pp. 21–30. (Paper=127, Status=F, Phase=6)
[[128]] H. Olague, L. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes," *Software Engineering, IEEE Transactions on*, vol. 33, no. 6, pp. 402 –419, june 2007. (Paper=128, Status=F, Phase=2)
[[129]] A. Oral and A. Bener, "Defect prediction for embedded software," in *Computer and information sciences, 2007. iscis 2007. 22nd international symposium on*, nov. 2007, pp. 1 –6. (Paper=129, Status=F, Phase=2, Data=N)
[[130]] T. J. Ostrand and E. J. Weyuker, "The distribution of faults in a large industrial software system," *SIGSOFT Softw. Eng. Notes*, vol. 27, pp. 55–64, July 2002. (Paper=130, Status=F, Phase=1)
[[131]] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Locating where faults will be," in *Proceedings of the 2005 conference on Diversity in computing*, ser. TAPIA '05. New York, NY, USA: ACM, 2005, pp. 48–50. (Paper=131, Status=F, Phase=2)
[[132]] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Automating algorithms for the identification of fault-prone files," in *Proceedings of the 2007 international symposium on Software testing and analysis*, ser. ISST. NY, USA: ACM, 2007, pp. 219–227. (Paper=132, Status=F, Phase=4)
[[133]] T. J. Ostrand, E. J. Weyuker, and R. Bell, "Where the bugs are," in *ACM SIGSOFT Software Engineering Notes*, vol. 29. ACM, 2004, pp. 86–96. (Paper=133, Status=P)
[[134]] T. J. Ostrand, E. J. Weyuker, and R. Bell, "Predicting the location and number of faults in large software systems," *Software Engineering, IEEE Transactions on*, vol. 31, no. 4, pp. 340 – 355, april 2005. (Paper=134, Status=F, Phase=4)
[[135]] T. Ostrand, E. Weyuker, and R. Bell, "Programmer-based fault prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*. ACM, 2010, pp. 1–10. (Paper=135, Status=P)
[[136]] T. Ostrand, E. Weyuker, R. Bell, and R. Ostrand, "A different view of fault prediction," in *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, vol. 2, july 2005, pp. 3 – 4 Vol. 1. (Paper=136, Status=F, Phase=1)
[[137]] F. Padberg, T. Ragg, and R. Schoknecht, "Using machine learning for estimating the defect content after an inspection," *Software Engineering, IEEE Transactions on*, vol. 30, no. 1, pp. 17 – 28, jan. 2004. (Paper=137, Status=F, Phase=2)
[[138]] G. Pai and J. Dugan, "Empirical analysis of software fault content and fault proneness using bayesian methods," *Software Engineering, IEEE Transactions on*, vol. 33, no. 10, pp. 675 –686, oct. 2007. (Paper=138, Status=F, Phase=2, Data=N)
[[139]] A. K. Pandey and N. K. Goyal, "Test effort optimization by prediction and ranking of fault-prone software modules," in *Reliability, Safety and Hazard (ICRESH), 2010 2nd International Conference on*. accept, 2010, pp. 136–142. (Paper=139, Status=F, Phase=2, Data=N)
[[140]] L. Pelayo and S. Dick, "Applying novel resampling strategies to software defect prediction," in *Fuzzy Information Processing Society, 2007. NAFIPS '07. Annual Meeting of the North American*, june 2007, pp. 69 –72. (Paper=140, Status=F, Phase=2, Data=N)
[[141]] P. Pendharkar, "Exhaustive and heuristic search approaches for learning a software defect prediction model," *Engineering Applications of Artificial Intelligence*, vol. 23, no. 1, pp. 34–40, 2010. (Paper=141, Status=F, Phase=6, Data=N)
[[142]] H. Peng and Z. Jie, "Predicting defect-prone software modules at different logical levels," in *Research Challenges in Computer Science, 2009. ICRCCS '09. International Conference on*. accept, 2009, pp. 37–40. (Paper=142, Status=F, Phase=2, Data=N)
[[143]] M. Pighin and A. Marzona, "An empirical analysis of fault persistence through software releases," in *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on*, sept.-1 oct. 2003, pp. 206 – 212. (Paper=143, Status=F, Phase=1)
[[144]] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures? " in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2008, pp. 2–12. (Paper=144, Status=F, Phase=2)
[[145]] R. Ramler, S. Larndorfer, and T. Natschlager, "What software repositories should be mined for defect predictors? " in *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on*. accept, 2009, pp. 181–187. (Paper=145, Status=F, Phase=4)
[[146]] Z. Rana, S. Shamail, and M. Awais, "Ineffectiveness of use of software science metrics as predictors of defects in object oriented software," in *Software Engineering, 2009. WCSE '09. WRI World Congress on*, vol. 4, may 2009, pp. 3 –7. (Paper=146, Status=F, Phase=1, Data=N)
[[147]] J. Ratzinger, T. Sigmund, and H. C. Gall, "On the relation of refactorings and software defect prediction," in *Proceedings of the 2008 international working conference on Mining software repositories*, ser. MSR '08, NY ACM, 2008, pp. 35–38. (Paper=147, Status=F, Phase=4)
[[148]] M. Reformat, *A Fuzzy-Based Meta-model for Reasoning about the Number of Software Defects*, ser. Lecture Notes in Computer Science. Springer, Lecture Notes in Computer Science, 2003, vol. 2715. (Paper=148, Status=F, Phase=2)
[[149]] D. Rodriguez, R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz, "Detecting fault modules applying feature selection to classifiers," in *Information Reuse and Integration. IRI. IEEE International Conference on*, aug. 2007, pp. 667 –672. (Paper=149, Status=F, Phase=2, Data=N)
[[150]] P. S. Sandhu, R. Goel, A. S. Brar, J. Kaur, and S. Anand, "A model for early prediction of faults in software systems," in *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conf on*, vol. 4, 2010, pp. 281–285. (Paper=150, Status=F, Phase=2, Data=N)
[[151]] P. S. Sandhu, M. Kaur, and A. Kaur, "A density based clustering approach for early detection of fault prone modules," in *Electronics and Information Engineering (ICEIE), International Conf On*, vol. 2, 2010, pp. V2–525–V2–530. (Paper=151, Status=F, Phase=2, Data=N)
[[152]] N. Schneidewind, "Investigation of logistic regression as a discriminant of software quality," in *Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International*, 2001, pp. 328 –337. (Paper=152, Status=F, Phase=2)

[[153]] A. Schröter, T. Zimmermann, R. Premraj, and A. Zeller, "If your bug database could talk," in *Proceedings of the 5th international symposium on empirical software engineering*, vol. 2, 2006, pp. 18–20. (Paper=153, Status=F, Phase=1)

[[154]] A. Schröter, T. Zimmermann, and A. Zeller, "Predicting component failures at design time," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. ACM, 2006, pp. 18–27. (Paper=154, Status=P)

[[155]] A. Schröter, T. Zimmermann, R. Premraj, and A. Zeller, "Where do bugs come from? " *SIGSOFT Softw. Eng. Notes*, vol. 31, pp. 1–2, November 2006. (Paper=155, Status=F, Phase=1)

[[156]] C. Seiffert, T. M. Khoshgoftaar, and J. V. Hulse, "Improving software-quality predictions with data sampling and boosting," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 39, no. 6, pp. 1283–1294, 2009. (Paper=156, Status=F, Phase=2, Data=N)

[[157]] N. Seliya, T. M. Khoshgoftaar, and J. Van Hulse, "Predicting faults in high assurance software," in *High-Assurance Systems Engineering (HASE), 2010 IEEE 12th International Symposium on*, 2010, pp. 26–34. (Paper=157, Status=F, Phase=2, Data=N)

[[158]] N. Seliya, T. Khoshgoftaar, and S. Zhong, "Analyzing software quality with limited fault-proneness defect data," in *High-Assurance Systems Engineering, 2005. HASE 2005. Ninth IEEE International Symposium on*, oct. 2005, pp. 89 –98. (Paper=158, Status=F, Phase=2, Data=N)

[[159]] R. Selvarani, T. Nair, and V. Prasad, "Estimation of defect proneness using design complexity measurements in object-oriented software," in *2009 International Conference on Signal Processing Systems*, may 2009, pp. 766 –770. (Paper=159, Status=F, Phase=1)

[[160]] R. Shatnawi and W. Li, "The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process," *Journal of Systems and Software*, vol. 81, no. 11, pp. 1868–1882, 2008. (Paper=160, Status=P)

[[161]] M. Sherriff, S. S. Heckman, M. Lake, and L. Williams, "Identifying fault-prone files using static analysis alerts through singular value decomposition," in *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, ser. CASCON '07. New York, NY, USA: ACM, 2007, pp. 276–279. (Paper=161, Status=F, Phase=2)

[[162]] M. Sherriff, N. Nagappan, L. Williams, and M. Vouk, "Early estimation of defect density using an in-process haskell metrics model," *SIGSOFT Softw. Eng. Notes*, vol. 30, pp. 1–6, May 2005. (Paper=162, Status=F, Phase=1)

[[163]] Y. Shin, R. M. Bell, T. J. Ostrand, and E. J. Weyuker, "Does calling structure information improve the accuracy of fault prediction? " in *Proceedings of the 6th International Working Conference on Mining Software Repositories, MSR 2009 (Co-located with ICSE)*, M. W. Godfrey and J. Whitehead, Eds. Vancouver, BC, Canada, May 16-17: IEEE, 2009, pp. 61–70. (Paper=163, Status=P)

[[164]] S. Shivaji, E. J. Whitehead, R. Akella, and K. Sunghun, "Reducing features to improve bug prediction," in *Automated Software Engineering, 2009. ASE '09. 24th IEEE/ACM International Conference on.* accept, 2009, pp. 600–604. (Paper=164, Status=P)

[[165]] P. Singh and S. Verma, "An investigation of the effect of discretization on defect prediction using static measures," in *Advances in Computing, Control, Telecommunication Technologies, ACT '09. International Conf on*, 2009, pp. 837–839. (Paper=165, Status=F, Phase=2)

[[166]] Y. Singh, A. Kaur, and R. Malhotra, "Predicting software fault proneness model using neural network," *Product-Focused Software Process Improvement*, vol. 5089, pp. 204–214, 2008. (Paper=166, Status=F, Phase=2, Data=N)

[[167]] Y. Singh, A. Kaur, and R. Malhotra, "Empirical validation of object-oriented metrics for predicting fault proneness models," *Software Quality Journal*, vol. 18, no. 1, pp. 3–35, 2010. (Paper=167, Status=F, Phase=2, Data=N)

[[168]] Q. Song, M. Shepperd, M. Cartwright, and C. Mair, "Software defect association mining and defect correction effort prediction," *Software Engineering, IEEE Transactions on*, vol. 32, no. 2, pp. 69 – 82, feb. 2006. (Paper=168, Status=F, Phase=2)

[[169]] C. Stringfellow and A. Andrews, "Deriving a fault architecture to guide testing," *Software Quality Journal*, vol. 10, no. 4, pp. 299–330, 2002. (Paper=169, Status=F, Phase=1)

[[170]] G. Succi, W. Pedrycz, M. Stefanovic, and J. Miller, "Practical assessment of the models for identification of defect-prone classes in object-oriented commercial systems using design metrics," *J of Systems and Software*, vol. 65, no. 1, pp1–12, 2003. (Paper=170, Status=F, Phase=1)

[[171]] M. D. M. Suffian and M. R. Abdullah, "Establishing a defect prediction model using a combination of product metrics as predictors via six sigma methodology," in *Information Technology (ITSim), International Symp in*, vol. 3, 2010, pp. 1087–1092. (Paper=171, Status=F, Phase=2)

[[172]] M. M. T. Thwin and T.-S. Quah, "Application of neural network for predicting software development faults using object-oriented design metrics," in *Neural Information Processing, 2002. ICONIP '02. Proceedings of the 9th International Conference on*, vol. 5, nov. 2002, pp. 2312 – 2316 vol.5. (Paper=172, Status=F, Phase=2)

[[173]] P. Tomaszewski, H. Grahn, and L. Lundberg, "A method for an accurate early prediction of faults in modified classes," in *Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference on*, sept. 2006, pp. 487 –496. (Paper=173, Status=F, Phase=2)

[[174]] A. Tosun and A. Bener, "Reducing false alarms in software defect prediction by decision threshold optimization," in *Empirical Software Engineering and Measurement, 2009. ESEM. 3rd International Symposium on*, 2009, pp477–480. (Paper=174, Status=F, Phase=2, Data=N)

[[175]] A. Tosun, B. Turhan, and A. Bener, "Practical considerations in deploying ai for defect prediction: a case study within the turkish telecommunication industry," in *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. ACM, 2009, p. 11. (Paper=175, Status=F, Phase=2, Data=N)

[[176]] A. Tosun, A. B. Bener, B. Turhan, and T. Menzies, "Practical considerations in deploying statistical methods for defect prediction: A case study within the turkish telecommunications industry," *Information and Software Technology*, vol. 52, no. 11, pp. 1242–1257, 2010. (Paper=176, Status=F, Phase=2, Data=N)

[[177]] B. Turhan and A. Bener, "A multivariate analysis of static code attributes for defect prediction," in *Quality Software, 2007. QSIC '07. Seventh International Conference on*, oct. 2007, pp. 231 –237. (Paper=177, Status=F, Phase=2, Data=N)

[[178]] B. Turhan, G. Kocak, and A. Bener, "Software defect prediction using call graph based ranking (cgbr) framework," in *Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference*, sept. 2008, pp. 191 –198. (Paper=178, Status=F, Phase=2)

[[179]] B. Turhan, G. Kocak, and A. Bener, "Data mining source code for locating software bugs: A case study in telecommunication industry," *Expert Systems with Applications*, vol. 36, no. 6, pp. 9986–9990, 2009. (Paper=179, Status=F, Phase=2, Data=N)

[[180]] B. Turhan, A. B. Bener, and T. Menzies, "Regularities in learning defect predictors," in *Product-Focused Software Process Improvement, 11th International Conference, PROFES 2010*, M. A. Babar, M. Vierimaa, and M. Oivo, Eds., vol. 6156. Limerick, Ireland, June 21-23: Springer. series Lecture Notes in Business Information Processing, 2010, pp. 116–130. (Paper=180, Status=F, Phase=4, Data=N)

[[181]] B. Turhan, T. Menzies, A. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, 2009. (Paper=181, Status=F, Phase=2, Data=N)

[[182]] O. Vandecruys, D. Martens, B. Baesens, C. Mues, M. De Backer, and R. Haesen, "Mining software repositories for comprehensible software fault prediction models," *Journal of Systems and Software*, vol. 81, no. 5, pp. 823–839, 2008. (Paper=182, Status=F, Phase=2, Data=N)

[[183]] R. Vivanco, Y. Kamei, A. Monden, K. Matsumoto, and D. Jin, "Using search-based metric selection and oversampling to predict fault prone modules," in *Electrical and Computer Engineering (CCECE), 23rd Canadian Conf on*, 2010, pp1–6. (Paper=183, Status=F, Phase=2, Data=N)

[[184]] D. Wahyudin, A. Schatten, D. Winkler, A. M. Tjoa, and S. Biffl, "Defect prediction using combined product and project metrics - a case study from the open source "apache" myfaces project family," in *Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference*. accept, 2008, pp. 207–215. (Paper=184, Status=F, Phase=1)
[[185]] T. Wang and W.-h. Li, "Naive bayes software defect prediction model," in *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*. accept, 2010, pp. 1–4. (Paper=185, Status=F, Phase=2, Data=N)
[[186]] W. Wei, D. Xuan, L. Chunping, and W. Hui, "A novel evaluation method for defect prediction in software systems," in *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*, 2010, pp. 1–5. (Paper=186, Status=F, Phase=1)
[[187]] Y. Weimin and L. Longshu, "A rough set model for software defect prediction," in *Intelligent Computation Technology and Automation (ICICTA), 2008 International Conference on*, vol. 1, 2008, pp. 747–751. (Paper=187, Status=F, Phase=2, Data=N)
[[188]] E. Weyuker, T. Ostrand, and R. Bell, "Using developer information as a factor for fault prediction," in *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, may 2007, p. 8. (Paper=188, Status=F, Phase=2)
[[189]] E. Weyuker, T. Ostrand, and R. Bell, "Comparing negative binomial and recursive partitioning models for fault prediction," in *Proceedings of the 4th international workshop on Predictor models in software engineering*. ACM, 2008, pp. 3–10. (Paper=189, Status=F, Phase=2)
[[190]] E. Weyuker, T. Ostrand, and R. Bell, "Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models," *Empirical Software Engineering*, vol. 13, no. 5, pp. 539–559, 2008. (Paper=190, Status=P)
[[191]] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, "Comparing the effectiveness of several modeling methods for fault prediction," *Empirical Software Engineering*, vol. 15, no. 3, pp. 277–295, 2010. (Paper=191, Status=F, Phase=2)
[[192]] C. Wohlin, M. Host, and M. Ohlsson, "Understanding the sources of software defects: a filtering approach," in *Program Comprehension, 2000. Proceedings. IWPC 2000. 8th International Workshop on*, 2000, pp. 9 –17. (Paper=192, Status=F, Phase=1)
[[193]] W. Wong, J. Horgan, M. Syring, W. Zage, and D. Zage, "Applying design metrics to predict fault-proneness: a case study on a large-scale software system," *Software: Practice and Experience*, vol. 30, no. 14, pp. 1587–1608, 2000. (Paper=193, Status=F, Phase=2)
[[194]] Z. Xu, T. Khoshgoftaar, and E. Allen, "Prediction of software faults using fuzzy nonlinear regression modeling," in *High Assurance Systems Engineering, 2000, Fifth IEEE International Symposim on. HASE 2000*, 2000, pp. 281 –290. (Paper=194, Status=F, Phase=2)
[[195]] B. Yang, L. Yao, and H.-Z. Huang, "Early software quality prediction based on a fuzzy neural network model," in *Natural Computation, 2007. ICNC 2007. Third International Conference on*, vol. 1, aug. 2007, pp. 760 –764. (Paper=195, Status=F, Phase=2)
[[196]] L. Yi, T. M. Khoshgoftaar, and N. Seliya, "Evolutionary optimization of software quality modeling with multiple repositories," *Software Engineering, IEEE Transactions on*, vol. 36, no. 6, pp. 852–864, 2010. (Paper=196, Status=F, Phase=2, Data=N)
[[197]] H. Youngki, B. Jongmoon, K. In-Young, and C. Ho-Jin, "A value-added predictive defect type distribution model based on project characteristics," in *Computer and Information Science, 2008. ICIS 08. Seventh IEEE/ACIS International Conference on*, 2008, pp. 469–474. (Paper=197, Status=F, Phase=2)
[[198]] P. Yu, T. Systa, and H. Muller, "Predicting fault-proneness using oo metrics. an industrial case study," in *Software Maintenance and Reengineering, 2002. Proceedings. Sixth European Conference on*, 2002, pp. 99 –107. (Paper=198, Status=F, Phase=1)
[[199]] X. Yuan, T. Khoshgoftaar, E. Allen, and K. Ganesan, "An application of fuzzy clustering to software quality prediction," in *Application-Specific Systems and Software Engineering Technology. Procs. 3rd IEEE Symposium on*, 2000, pp85–90. (Paper=199, Status=F, Phase=2)
[[200]] H. Zhang, A. Nelson, and T. Menzies, "On the value of learning from defect dense components for software defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering, PROMISE 2010*, T. Menzies and G. Koru, Eds. Timisoara, Romania, September 12-13, 2010: ACM, 2010, p. 14. (Paper=200, Status=F, Phase=2, Data=N)
[[201]] S. Zhong, T. Khoshgoftaar, and N. Seliya, "Unsupervised learning for expert-based software quality estimation," in *High Assurance Systems Engineering, 2004. Proceedings. Eighth IEEE International Symposium on*, march 2004, pp149–155. (Paper=201, Status=F, Phase=2, Data=N)
[[202]] Y. Zhou and H. Leung, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults," *Software Engineering, IEEE Transactions on*, vol. 32, no. 10, pp. 771 –789, oct. 2006. (Paper=202, Status=F, Phase=2, Data=N)
[[203]] Y. Zhou, B. Xu, and H. Leung, "On the ability of complexity metrics to predict fault-prone classes in object-oriented systems," *Journal of Systems and Software*, vol. 83, no. 4, pp. 660–674, 2010. (Paper=203, Status=P)
[[204]] T. Zimmermann and N. Nagappan, "Predicting subsystem failures using dependency graph complexities," in *Software Reliability, 2007. ISSRE '07. The 18th IEEE International Symposium on*, nov. 2007, pp. 227 –236. (Paper=204, Status=F, Phase=2)
[[205]] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on*, 2008, pp. 531–540. (Paper=205, Status=F, Phase=4)
[[206]] T. Zimmermann and N. Nagappan, "Predicting defects with program dependencies," in *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, 2009, pp. 435–438. (Paper=206, Status=F, Phase=2)
[[207]] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, may 2007, p. 9. (Paper=207, Status=F, Phase=2)
[[208]] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ser. ESEC/FSE '09. New York, 2009, pp. 91–100. (Paper=208, Status=F, Phase=4)

Appendix A: Search string

The following search string was used in our searches:

(Fault* OR bug* OR defect* OR errors OR corrections OR corrective OR fix*) *in title only*
AND (Software) *anywhere in study*

Appendix B: Conferences and Journals manually searched

Conference manually searched	Journals manually searched
International Conference on Software Engineering (ICSE)	IEEE Transactions of Software Engineering
International Conference on Software Maintenance (ICSM)	Journal of Systems and Software
IEEE Int'l Working Conference on Source Code Analysis and Manipulation (SCAM)	Journal of Empirical Software Engineering
International Conference on Automated Software Engineering	Software Quality Journal
IEEE International Symposium and Workshop on Engineering of Computer Based Systems	Information & Software Technology
International Symposium on Automated Analysis-driven Debugging	
International Symposium on Software Testing and Analysis (ISSTA)	
International Symposium on Software Reliability Engineering	
ACM SIGPLAN Conference on Programming language Design and Implementation	
Int'l Workshop on Mining Software Repositories	
Empirical Software Engineering & Measurement	
PROMISE	
Foundations of Software Engineering	

Appendix C: Additional assessment criteria

Data quality criteria. The efficacy of the predictions made by a model is determined by the quality of the data on which the model was built. Leibchen and Shepperd [21] report that many studies do not seem to consider the quality of the data they use. Many fault prediction models are based on machine learning where it has been shown that lack of data cleaning may compromise the predictions obtained [25]. The criteria shown in Table C1 are based on Gray et al [25], Song et al [[168]], Wohlin et al [[192]], Zhiwei et al [[194]] and Boetticher [[19]].

Table C1. Additional data quality criteria

Data quality criteria	Criteria definitions	Why the criteria is important
Has any data cleaning been done?	An indication that the quality of the data has been considered is necessary and that any data cleaning needed has been addressed e.g. missing values handled, outliers and errorful data been removed.	Data sets are often noisy. They often contain outliers and missing values that can skew results (the impact of this depends on the analysis methods used). Our confidence in the predictions made by a model is impacted by the quality of the data used while building the model. Few studies have cleaned their data and so we did not apply this criterion.
Have repeated attributes been removed?	An indication that the impact of repeated attributes has been considered should be given. For example machine learning studies should mention attribute selection while other studies should consider, for example Principal Component Analysis.	Repeated attributes and related attributes have been shown to bias the outcomes of models. Confidence is affected in the predictions of studies which have not considered the impact of repeated/related attributes. Few studies have considered repeated attributes and so we did not apply this criterion.

Predictive performance criteria. Measuring the predictive performance of a model is an essential part of demonstrating the usefulness of that model. Measuring model performance is complex and there are many ways in which the performance of a model may be measured. Furthermore the value of measures varies according to context. For example, safety critical system developers may want models that identify as many faults as possible, accepting the cost of false alarms. Whereas business system developers may want models which do not generate many false alarms, as testing effort is short to ensure the timely release of a product, at the cost of missing some faults. Appendix D reports the principles of predictive performance measurement and provides the basis of our performance measurement criteria. Table C2 shows our predictive performance measurement criteria.

Table C2. Additional predictive performance measurement criteria

Measurement criteria	Criteria definitions	Why the criteria is important
Have imbalanced data sets been accounted for, sufficient to enable confidence in predictive performance?	No one approach adequately accounts for imbalanced data in every circumstance. See Appendix F for an overview of the approaches available. Each study was assessed on a case-by-case basis according to the specific model reported.	See Appendix F for an overview of the importance of dealing with imbalanced data. There is no one best way of accounting for data imbalance and it was thus difficult to identify a precise enough criterion to apply consistently across studies. In addition there remains significant debate on data imbalance (see [22], [[179]], [23], [24]).
Has predictive performance been reported appropriately?	For each model reporting categorical results a study should report: - A confusion matrix - Area Under the Curve (AUC) For each model reporting continuous results a study choosing to report measures of error should not report only Mean Squared Error. Average Relative Error should also be reported. Or else results based on Chi Square should be reported.	The particular set of performance measures reported by studies can make it difficult to understand how a model performs overall in terms of correct and incorrect predictions. Confusion matrix constructs form the basis of most other ways of reporting predictive performance. Reporting the confusion matrix (possibly in addition to other measures reported by studies) would allow subsequent analysis of performance in ways other than those preferred by the model developer. Menzies et al [[114]] suggests a useful way in which data from multiple confusion matrices may be effectively reported. AUC performance data is reported to be an effective way in which to compare the ability of a modelling technique to cope with different datasets (Lessmannn et al [[97]]). Reporting AUC means that models using different datasets can then be meta-analysed. This would enable a much richer understanding of the abilities of particular modelling techniques. Ideally data for the whole ROC curve would be given by each study. It is impractical to report this amount of data and would require the use of on-line data stores. Mean Squared Error (MSE) generates results that can only be interpreted within the data set from which they originated as the measurement scales used dictate the size of the error. MSE limits the comparability of results across other data sets (this is why [[31]] reports Chi Square results as this normalises the error). Chi Square or Average Relative Error should be used instead. Only a small number of models currently report confusion matrix and AUC data. Consequently, applying this criterion was untenable as so few studies would have passed. Similarly MSE is a poorly understood measure and its limitations are not widely reported and so the current application of this requirement is not tenable.

Appendix D: The principles of predictive performance measurement.

This overview of measuring predictive performance is based on work by Arisholm et al [2], Ostrand and Weyuker [26], Jiang et al [[61]] and Lessmannn et al [[97]]. The measurement of predictive performance is often based on the analysis of data in a confusion matrix (shown in Table Dl and explained further in Table D2). This matrix reports how the model classified the different fault categories compared to their actual classification (predicted versus observed). Many performance measures are related to components of the confusion matrix shown in Table D2. Confusion matrix based measures are most relevant to fault prediction models producing categorical outputs, though continuous outputs can be converted to categorical outputs and analysed in terms of a confusion matrix.

	Predicted defective	Predicted defect free
Observed defective	True Positive (TP)	False Negative (FN)
Observed defect free	False Positive (FP)	True Negative (TN)

Table D1. A confusion matrix

Construct	Also known as	Description
False Positive	FP, and Type I Error	Classifies non faulty unit as faulty
False Negative	FN, and Type II Error	Classifies faulty unit as not faulty
True Positive	TP	Correctly classified as faulty
True Negative	TN	Correctly classified as non-faulty

Table D2. Confusion matrix based performance indicator

Composite performance measures can be calculated by combining values from the confusion matrix (see Table D3). 'Recall' (otherwise known as the true positive rate, probability of detection (pd) or sensitivity) describes the proportion of faulty code units (usually files, modules or packages) correctly predicted as such, while 'precision' describes how reliable a prediction is in terms of what proportion of code predicted as faulty actually was faulty. Both are important when modeling with imbalanced data, but there is a trade-off between these two measures [[61]]. An additional composite measure is the false positive rate (pf) which describes the proportion of erroneous defective predictions. Thus, the optimal classifier would achieve a pd of 1, precision of 1 and a pf of 0. The performance measure balance combines pd and pf. A high balance value (near 1) is achieved with a high pd and low pf. Balancing can also be adjusted to factor in the cost of false alarms which typically do not result in fault fixes. When the combinations of pd and pf are plotted they produce a Receiver Operator Curve (ROC). This gives a range of balance figures, and it is usual to report the area under the curve (AUC) as varying between 0 and 1, with 1 being an ideal value and 0.5 being achieved by a model predicting randomly with balanced data.

Construct	Defined as	Description
Recall pd (probability of detection) Sensitivity True positive rate	TP / (TP + FN)	Proportion of faulty units correctly classified
Precision	TP / (TP + FP)	Proportion of units predicted as faulty
pf (probability of false alarm) False positive rate	FP / (FP + TN)	Proportion of non-faulty units incorrectly classified
Specificity	TN / (TN + FP)	Proportion of correctly classified non faulty units
f-measure	(2 x Recall x Precision) / (Recall + Precision)	Most commonly defined as the harmonic mean of precision and recall
Accuracy	(TN + TP) / (TN + FN + FP + TP)	Proportion of correctly classified units
Mis-classification rate Error-rate	1-accuracy	Proportion of incorrectly classified units
Balance	$balance = 1 - \frac{\sqrt{(0-pf)^2 + (1-pd)^2}}{\sqrt{2}}$	Combines *pf* and *pd* into one measure and is most commonly defined as the distance from the ROC 'sweet spot' (where pd=1, pf=0).
Receiver operating characteristic (ROC) curve		A graphical plot of the sensitivity (or pd) vs. 1 – specificity (or pf) for a binary classification system where its discrimination threshold is varied

Table D3. Composite performance measures

Table D4 shows other ways in which the performance of a model can be measured. Such measures are usually used in models that produce continuous or ranking results.

Measure	Constructs and Definitions
Error measures	Average residual error, relative error, relative square error, standard error of estimate, root mean squared error, median relative error, mean square error, mean absolute error, mean absolute relative error, error rate.
Significance of difference between predicted and observed	Spearmans, Pearsons, Chi Square

Table D4. Performance indicators defined

Appendix E: Calculating precision, recall and f-measure for categorical studies (reported in [27])

Many studies report *precision* and *recall*, but others report *pd* and *pf*. If we are to compare the results we need to convert the results of one paper into the performance measures reported by the other paper. In this case we want to report everything in terms of *precision* and *recall*. We chose these measures as defect prediction data sets are often highly imbalanced (Zhang and Zhang [22] and Gray et al [18]). When trying to compare the results of one paper with the results of another paper, it may be necessary to reconstruct a form of the Confusion Matrix (see Table D1 in Appendix D) where the values are not the sums of instances, but the frequency of each instance. This is possible in many cases when the distribution of the classes is also reported. To do this we need to know the frequency of the true class *d*, where

$$1 = TP + TN + FP + FN \tag{1}$$

$$d = TP + FN \tag{2}$$

It then becomes possible to calculate *TP* (True Positive), *FP* (False Positive), *TN* (True Negative) and *FN* (False Negative) as follows:

Given *pf* and *d*

$$TN = (1 - d)(1 - pf) \tag{3}$$

$$PF = (1 - d)pf \tag{4}$$

Given *pd*(*Recall*(*r*)) and *d*

$$TP = d(r) \tag{5}$$

$$FN = d(1 - r) \tag{6}$$

Given *FNR*(*TypeII*(*t*2)), *pf* and *d* we already have (1), (3) and (4)

$$FN = \frac{pf(1 - d)t2}{1 - t2} \tag{7}$$

$$TP = 1 - FN - TN - TP \tag{8}$$

Given *precision*(*p*), *recall*(*r*) and *d* we already have (1), (5) and (6)

$$FP = \frac{d.r(1 - p)}{p} \tag{9}$$

$$TN = 1 - FP - FN - TP \tag{10}$$

In some cases *d* is not available but more performance measures are provided.

Given *errorrate*(*er*), *FNR*(*TypeII*(*t*2)) and *pf*

$$d = 1 - \frac{er(1 - t2)}{pf} \tag{11}$$

which can then be used with (3), (4), (7) and (8)

Given *precision*(*p*), *recall*(*r*) and *accuracy*(*a*)

$$d = \frac{p(1 - a)}{P - 2pr + r} \tag{12}$$

which can then be used with (5), (6), (9) and (10)

F-Measure / F-Score = 2.recall.precision/(recall + precision)

An example using paper [[83]]. The following values were extracted from [[83]]

er = 0.3127 *pf* = 0.3134 *t*2 = 0.2826
We compute: *d* = 0.2842
Giving: *FN* = 0.0884 *TN* = 0.4915 *FP* = 0.2243 *TP* = 0.1958
Finally: *precision* = 0.4661 *recall* = 0.6891 *f-Measure* = 0.5561

Appendix F: The Class Imbalance Problem

Substantially imbalanced data sets are commonly used in binary fault prediction studies (i.e. there are usually many more non-faulty units than faulty units) [28] and [22]. An extreme example of this is seen in NASA data set PC2, which has only 0.4% of data points belonging to the faulty class (23 out of 5589 data points). This distribution of faulty and non-faulty units - known as the class distribution, should be taken into account during any binary fault prediction task. This is because imbalanced data can strongly influence both: the training of a classification model, and the suitability of classifier performance metrics.

When training a classifier using imbalanced data, an algorithm can struggle to learn from the minority class. This is typically due to an insufficient quantity of minority class data. The most common symptom when this occurs is for a classifier to predict all data points as belonging to the majority class, which is of little practical worth. To avoid this happening, various approaches can be used, and are typically based around training-set sampling and/or learning algorithm optimisation. Note that these techniques are entirely optional, and may not be necessary. This is because learning techniques vary in their sensitivity to imbalanced data. For example: C4.5 decision trees have been reported to struggle with imbalanced data [14], whereas fuzzy based classifiers have been reported to perform robustly regardless of class distribution [29].

Sampling methods involve the manipulation of training data in order to reduce the level of imbalance, and therefore alleviate the problems associated with learning from imbalanced data. Under-sampling methods involve reducing the size of the majority class, whereas over-sampling methods involve increasing the size of the minority class. Such techniques have been reported to be useful [30], however they do suffer from drawbacks. With under-sampling methods, the main problem is deciding which majority class data points should be removed. With over-sampling methods, there is a risk of the learning algorithm over-fitting the over-sampled data. This will probably result in good training data performance, but low performance when the classifier is presented with unseen data (data independent from that used during training) [30].

Many learning algorithms can have their various parameters adjusted in order to boost performance on imbalanced data. This can be very effective, as many algorithms by default assume an equal class distribution during training. By increasing the misclassification cost of the minority class, it is possible to construct models that are better suited to imbalanced domains. The drawback of such methods is that it can be difficult and/or time consuming to approximate appropriate misclassification costs.

Additional problems caused by imbalanced data are that the selection of appropriate classifier performance measures becomes more difficult. This is because measures which favour the majority class (such as accuracy and error rate) are no longer sufficient [30]. More appropriate measures in imbalanced domains include: precision, recall, f-measure (see Appendix D) and g-mean [30].

In contrast to the training data, the balance of test data should representative of that which will be encountered in the systems on which the model will used. This is usually imbalanced as there are likely to be more non-faulty that faulty units in such data sets.

There remains significant debate on data imbalance in fault prediction (see [22], [[179]], [23], [24]).

Appendix G. Interpreting the Box Plots

The box represents 50% of the population. An outlier is considered to be any item which is more than 1.5 times the size of the box away from the top or bottom of the box.