A Systematic Review of Fault Prediction approaches used in Software Engineering

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

Tracy Hall
Brunel University

David Bowes
University of Hertfordshire

David Gray
University of Hertfordshire

Steve Counsell
Brunel University

Sue Black
University College London

Contact

Address	Lero International Science Centre 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-2010-04

A Systematic Review of Fault Prediction Approaches used in Software Engineering

Sarah Beecham[a], Tracy Hall[a], David Bowes[c], David Gray[c], Steve Counsell[a], Sue Black[d]

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

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

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

[d]University College London, Gower Street, London, WC1E 6BT UK

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

Abstract

BACKGROUND – The accurate prediction of where faults are likely to occur in code is important because it can help direct test effort, reduce costs and improve the quality of software.
OBJECTIVE – To summarise and analyse the published fault prediction studies in order to identify approaches used to build, measure and validate the performance of fault prediction models.
METHOD – A systematic literature review of fault prediction in 148 studies published from January 2000 to December 2009. Studies are classified in terms of their context, the variables and methods used to build models as well as how the performance of a model is measured and validated.
RESULTS – An increasing number of studies use machine learning approaches to predict where faults are likely to occur in code. These use a wide variety of methods to build models. Fault prediction models are based mainly on static code metrics, change data and previous fault data. The performance of models is measured in a range of ways that makes cross comparison very complex. The external validation of models is rarely demonstrated. Models reporting very high performance measures (eg. over 90%) need to be treated with particular caution.
CONCLUSION – The literature in this area is just beginning to mature with a small but growing number of studies appearing that transparently report models built using rigorous techniques; the performance of these models have been measured in credible ways. However, many studies do not present their methods clearly enough to enable the performance of their models to be convincingly demonstrated. A more standardised way of building and reporting on such performance is needed before potential model users can confidently evaluate these studies.

1. Introduction

This systematic literature review (SLR) aims to identify and analyse the models used to predict faults in source code in papers published over 10 years (between January 2000 and December 2009). The output from the review is a summary and evaluation of how studies have developed, used and validated models for fault prediction in software engineering. Many important aspects of model construction are analysed and a rigorous starting point in the area is provided for future researchers. This paper also provides a comprehensive entry point for practitioners preparing to use fault prediction models.

Fault prediction is a complex area of research and the subject of many previous studies. Software practitioners and researchers have explored various ways of predicting where faults are likely to occur in software to varying degrees of success. These studies typically produce fault prediction models that allow software engineers to focus development activities on fault-prone code, thereby improving software quality and making better use of resources. However, it

is difficult to get an overall picture of the current state of fault prediction, given the disparate nature of the individual models published.

Two previous reviews of the area have been performed ([9] and [4])[1]. Our review differs from these reviews in the following ways:

- *Timeframes.* Our review is the most contemporary as it includes studies published from 2000-2009. Fenton and Neil conducted a critical review of software fault prediction research up to 1999 [9]. Catal and Diri's [4] review covers work published between 1990 and 2007.
- *Systematic approach.* We follow Kitchenham's [12] 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 [10] methodology. Fenton and Neil did not apply the systematic approach introduced by Kitchenham [12] 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. Consequently, we analyse many more papers and include 74 papers which do not appear in Catal and Diri's review. Beecham et al [3] reports a paper-by-paper analysis of our SRL compared to Catal and Diri's.
- *Analysis.* We provide a more detailed analysis of each paper. Catal and Diri have 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 how the performance of models is measured and how models are validated.

This paper is organised as follows. In the next section, we provide some background to fault prediction. Section 3 presents our systematic literature review methodology. Section 4 contains the results of the papers we reviewed. A discussion of our results is presented in Section 5. Section 6 concludes the study.

2. Fault prediction modelling

Fault[2] prediction modelling has become a popular method for the early identification of fault-prone code. It is now the topic of a large body of software engineering literature. In this section we discuss factors which underpin the development of a prediction model likely and present basic recommendations relating to principles of measuring the performance of such models.

2.1 Developing a fault prediction model

Developing a reliable fault prediction model requires a number of model building concepts to be appropriately addressed. These concepts must be taken into account when reviewing models of fault prediction. These essential model building concepts include:

1. *Predictor or independent variables*. These are usually metrics based on software artefacts, such as static code metrics or change data, and are usually features that enable

[1] 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 which is located after the main reference list.

[2] 'Fault' is used interchangeably in this study with the terms 'defect' or 'bug' to mean a fault in software code.

some degree of fault prediction. This review is scoped to those models using independent variables based on units of code, such as files, classes or modules of code.

2. *Output or dependent variables.* The output from the model is a prediction of fault proneness in terms of faulty versus non-faulty code units. This output typically takes the form of either categorical or continuous output variables. Categorical outputs classify code units as either faulty or non-faulty. Continuous outputs usually provide the number of faults in a code unit[3].

3. *Modelling methods.* One or more modelling methods are used to explore the relationship between the predictor variables (or independent variables) and the outputs (or dependent variables). These methods may be, for example, types of statistic like regression or machine learning.

2.2 Measuring the performance of prediction models

Measuring the predictive performance of models is an essential part of model development and subject to on-going debate in the literature. We base the following overview of performance measures on work by Arisholm et al [2], Ostrand & Weyuker [15] and Lessman et al [[51]].

Reporting performance is often based on the analysis of data in a confusion matrix as shown in Table l and explained further in Table 2. 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 2. These components can be either within a confusion matrix or used individually. Confusion matrix measures of performance 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 1. 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 2. Confusion matrix based performance indicators

Composite performance measures can be calculated by combining values from the confusion matrix (see Table 3).These measures are most suitable when using imbalanced data as they are proportional to the class distribution (where code is viewed in two classes – faulty or non

[3] This binary division into continuous or categorical oversimplifies model outputs as, for example the output of a logistic regression is a probability of faultiness and it is only the cut-off value used that converts this into a categorical classification [[51]].

faulty). 'Recall' (otherwise known as the true positive rate, probability of detection (pd) or sensitivity) describes the proportion of defective code correctly predicted as such, while 'Precision' describes how reliable a defective prediction is, or, more specifically, what proportion of code predicted as defective actually was faulty. Both are important when modeling with imbalanced data, but there is a trade-off between these two measures [[146]]. Furthermore, particular requirements may favour one measure over another. For example, safety critical systems may be more interested in recall in order to catch as many defects as possible; business systems on the other hand may be more interested in precision. To remain competitive, business system development cannot afford to spend lots of time on testing as they need to release their product while business demand is high. 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.

Some machine learners have parameters that may be altered (e.g. a decision threshold) that results in different combinations of pd and pf. 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. However, some learners produce low AUC values but when tuned can produce high balance values.

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 3. Composite performance measures

Table 4 shows the other ways in which the performance of a model can be measured. Such measures can be used in models that produce either categorical or continuous outputs. Fault prediction studies often report several different performance indicators as the tools commonly used (e.g. Weka[4]) produce them automatically. However, the efficacy of the measure used depends on many aspects of the study and must be interpreted within the wider context of the study. Correct interpretation of model performance depends on many factors and requires a good understanding of data that contextualises the study. Often such context data is not reported or is presented ambiguously. Data balance (class distribution) in particular makes a significant difference to how some performance measures should be interpreted.

Measure	Constructs and Definitions
Completeness or effectiveness	% of faulty classes detected, proportion of total faults detected
Correctness	% of classes correctly predicted as faulty
Descriptive statistics	% of bugs found in system, Standard Dev, Mean, t-test, ranking, variance, z
Error measures	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.
Regression coefficients	Regression - logistic, Regression R2 (linear) - incl derived co-efficients, Regression R2 (nonlinear), cubic r2
Significance	Usually described in terms of p-value,for example $p \leq 0.05$

Table 4. Performance indicators defined

With the use of some performance indicators (in particular accuracy and error rate) data balancing (or class distribution) can be fundamental to developing reliable prediction models for categorical outputs. Most initial sets of fault data are significantly imbalanced with many more non faulty code units occurring in the data set than faulty units. Without model developers deliberately using over/under sampling methods (e.g. [6]) to balance the two classes of data, model predictions may be unreliable. In particular, machine learning (ML) classifiers are often unable to detect faulty modules since the majority of modules they learn from tend to be fault-free. Sun et al [18] note in their study using Support Vector Machines that "since the dataset is imbalanced, the [classification approach] will be expected to over-predict the majority class". Studies reporting model performance in terms of accuracy and which pay no regard to data balance, may report a significantly inflated model's performance. However discussion continues about the interplay between data balance and model performance indicators (see [[105]], [[113]], [3], [11]).

Demonstrating the validity of fault prediction models is essential. In the context of gaining insight into how well a defect predictor could perform on future projects, a model is only valid if it is tested on unseen data (i.e. data that was not used during the training process) [[62]]. The holdout validation technique is one way of demonstrating the internal validity of models. It is most relevant to machine learning approaches, but can also be applied to improve the robustness of results in statistical studies. With holdout experiments, the original data set is split into two groups: {training set, test set}. The model is developed using the training set and its

[4] Available at: http://www.cs.waikato.ac.nz/~ml/weka/index_downloading.html

performance is then assessed on the test set. As with all experiments, a result can occur by chance due to the way the data has been split. To overcome this an n > 1 fold cross validation process can be used, where the data is split into *n* groups {g1..g*n*}. For 10-fold cross validation, ten experiments are carried out with each of the groups being used as the test set in one of the ten runs; all other groups combined are thus used as the training set. Cross-validation is an extension to the holdout validation technique and is commonly used in machine learning studies. However cross-validation does not address the external validity of models and does not demonstrate how well a model will perform outside its immediate environment. This must be demonstrated by applying a model across systems, programs and with varying development groups [13].

3: Methodology

We take a systematic approach to reviewing the literature on the prediction of faults in code. Systematic literature reviews (SLR) are well established in medical research and increasingly in software engineering. We follow the systematic literature review approach identified by Kitchenham [12].

3.1 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 allows studies to be understood and interpreted by answering the research questions in Table 5.

Research Questions		Motivation
RQ1	What is the context of the fault prediction model?	This question allows us to understand the environment for which the prediction model was developed. We examine context primarily in terms of the origin of systems and the programming language that the model has been developed and tested on. This contextual information allows us to discuss the applicability and generalisability of models.
RQ2	What variables have been used in fault prediction models?	This question identifies the independent and dependent variables of the model. The answer shows the range of predictors of fault proneness used in models as well as the form that these take. We can also report on how thoroughly some variables are investigated compared to others.
RQ3	What modelling methods have been used in the development of fault prediction models?	This question identifies whether models are based on regression, machine learning or other approaches. The answer to this question allows us to discuss the popularity and effective use of particular modelling methods.
RQ4	How do studies measure the performance of their models?	Understanding how prediction studies measure and validate model performance gives an indication of how confident we can be in these results. We analyse several aspects of model performance (e.g. significance levels, data balance, accuracy) to discuss how well models are able to predict faults in code.

Table 5. The research questions addressed

It is important to note that a limitation of this study is that we do not report the performance of particular models. As there are multiple performance measures reported by both statistical models and machine learning models. Although we would like to be able to say that the best result for technique x produces a performance y, unfortunately there are too many factors which need to be taken into account when comparing studies even when they have reported the same performance measure to make this feasible.

3.2 Inclusion criteria

To be included in this review, a study must be reported in a complete paper published in English as either a Journal paper or Conference proceedings. The criteria for papers to be included in our SLR are based on the inclusion and exclusion criteria presented in Table 6.

Inclusion criteria	Exclusion criteria
A paper must be…	A paper must not be…
An empirical study	
Focused on fault prediction, estimation or validation	Highly specialised focused on, for example: testing, fault injection, inspections, reliability modelling, aspects, effort estimation, debugging, faults relating to memory leakage etc., embedded software, nano-computing, fault tolerance.
Use independent variables linked to outputs based on code	About the detection or localisation of existing individual known faults.
Faults in code is the main output (dependent variable)	Focused on the datasets/metrics used in the model rather than the prediction model

Table 6. 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.

3.3 Identification of papers

Our searches for papers were completed at the end of December 2009. Included papers were published between January 2000 and December 2009. There were three elements to our searches: an issue-by-issue manual reading of paper titles in relevant Journals and conferences; key word searching using search engines; identification of papers using references from included studies. The Journals and conferences shown in Appendix B were manually searched issue-by-issue. These were chosen as highly relevant software engineering publications found previously to be good sources of software engineering research [10].

Keyword searching was performed on the ACM Digital Library, IEEExplore and the ISI Web of Science. These search engines cover the vast majority of software engineering publications and the search string we used is given in Appendix A. We also included all papers within our timeframe from the Catal and Diri [5] review. Our initial searches omitted 23 of Catal and Diri's papers as their search term included the term 'quality'. We excluded this term from our searches as it generates a very high false positive rate.

Table 7 shows that our initial searches elicited 1,616 papers. The title and abstract of each was evaluated and 1,412 were rejected as not relevant to fault prediction. This process was validated using a randomly selected 80 papers from the initial 1,616. 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 looked at the remaining 202 papers in detail including checking for references to other relevant papers (we found a further 80 papers in this way). Data was extracted from each of these papers using the classification schemes discussed in the next section. During this process we found some papers that did not contain the data needed to answer our specific

research questions and these were also rejected. This process was validated by two researchers classifying all accepted papers according to the type of fault being predicted and the types of metrics used. A third researcher checked the classification of fault type. Finally the 23 extra papers from Catal and Diri's complementary review were also included. This process resulted in the 148 papers included in this review.

Selection Process	# of papers	Validation
Papers extracted from databases, conferences and journals	1,616	80 random papers independently classified by 3 researchers
Sift based on title and abstract	-1,414 rejected	Fair/good inter-rater agreement on first sift (k statistic test)
Full papers considered for review	202 primary 80 secondary	Each paper undergoes further checks when filling in data analysis forms, more papers rejected on the basis that they do not answer our research questions
Papers accepted for the review	**148 papers**	23 of these papers are sourced from Catal and Diri's review

Table 7. Paper selection and validation process

3.4 Analysis of papers

3.4.1 Developing classification schemes

We use several data classification schemes to facilitate our analysis. These are largely based on previous work on prediction modelling in software engineering and are discussed below.

Analysing the variables used in models. We use MacDonell and Shepperd's [13] work to motivate our analysis of the predictor/independent variable. Appendix D shows the more detailed coding scheme we devised to classify the static code independent variables studies used. We adapt Runeson et al's [17] scheme of output metrics to identify the two ways in which dependent variables (i.e. faults) may be classified, i.e., categorically or continuously.

Analysing the modelling methods used in models. We use two main categories of modeling methods: statistics and machine learning (this categorisation is largely based on Fenton and Neil [9]). Our finer grained analysis of statistical methods was developed by us to reflect the statistical methods used in studies and is shown in Appendix E. We base our finer grained analysis of the type of machine learning approaches used on Lessman et al's [[51]] classification of models (shown in Appendix F).

Analysing data balancing. We devised a classification scheme to understand how and if data imbalance had been dealt with by studies reporting categorical outputs (continuous models do not normally need to be as concerned about data balance). Our scheme was grounded in the studies we reviewed and is derived as follows: One researcher read through all papers reporting categorical models and extracted any approaches employed by the authors that might relate to how the studies address the problem of data imbalance. This list of approaches was given to a second researcher who used it to devise the categories shown in Appendix G.

Analysing the measurement of model performance. We analyse several aspects of model performance to discuss how well models predict faults in software. Our schemes for analysing the indicators used are grounded in the studies included in this review. We classify each study included in this review in terms of the performance indicators reported to have been used in the

papers. We analyse the performance of categorical studies (usually machine learning based studies) based on the application of the confusion matrix as shown in Appendix H. The performance of studies based on continuous data is usually based on statistics. We have analysed the statistics used in continuous studies as shown in Appendix I.

3.4.2 Extracting the detailed contents of each individual paper

Three researchers were involved in the extraction of data from studies using our classification schemes. Initially all three researchers extracted data from the same five studies. Their results were compared and any disagreements discussed in detail. As a result of this, the classification schemes were adjusted and the researchers' expertise on applying them improved. Data from a second set of five papers was then extracted by all three researchers using the classification schemes. Again, any disagreements were thoroughly discussed and a final adjustment was made to the classification schemes.

3.4.3 Presenting classification data

All papers included in this study are listed in a separate section of references and numbered 1-148. This special section of references appears after the main reference list. Each included paper reference has assigned to it a key showing the classifications extracted during the analysis phase of this review. The key provides data for each paper relating to how it has been classified for each of the data tables in this review (Tables 10-25). For example:

[[114]] Elish, K. O. and Elish, M. O. 2008. Predicting defect-prone software modules using support vector machines. *J. Syst. Softw.* 81, 5 (May. 2008), 649-660.
(Paper=114; T10=2; T11=1; T12=1: T13=3; T14=2; T15=1; T16=1; T18=4; T19=NA; T20/21=1,7; T22=2; T23=2; T24=n; T25=y)

The key for this paper shows that for Table 10 (data used) this paper is coded '2' (NASA data); for Table 11 (Language used) this paper is coded '1' (C/C++) etc.

References to individual papers are not shown in the tables in the Results Section as such tables would be unwieldy. This is similar to the approach taken by Jorgensen and Shepperd [10]. This utility also enables groups of prediction studies included in this report to be selected based on context, for example, all studies using C/C++ language and NASA data can be sourced from the reference section by searching for T10=2 and T11=1.

3.5 Threats to validity

Search terms: 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 papers. This term also generates a high number of false positive results. However, we might have missed some papers that use the term 'quality' as a synonym for 'defect' or 'fault', etc. We also omitted the term 'failure' from our search string as this generated papers predominately reporting on studies of software reliability in terms of the safety critical systems. Such studies of reliability predominately examine the dynamic behaviour of the system, and seldom look at the prediction of static code faults which is the focus of this review.

Search strategy: By applying our search terms to only the title of papers we may miss studies of fault prediction that do not use these terms in the title. Dieste et al [8] report that the effectiveness of searches can be improved if search terms are applied to both title and abstract fields. However, since we extend our searches to include papers cited in the included papers, we are confident that all key papers are included.

Extracted data: It is likely that we have coded some papers incorrectly. Studies in the field of fault prediction are complex and data from them is often hard to extract, comprehend and classify. Even after considerable discussion amongst the three researchers extracting data from the papers there was not always total agreement on how to code what was reported in a particular paper. Fitting a unique study within a fixed classification scheme is therefore open to some interpretation. Although we have spent many hours minimising misclassifications it is likely that some inconsistencies remain in the data. Any misclassification should be reported to the authors who will update the database that they maintain on this data. Furthermore, our assessment of model performance can only be based on what is explicitly reported by authors of studies. It could be that, for example, data has been balanced or particular performance tests conducted, but not reported. In addition, studies occasionally use terms such as 'accuracy' and 'precision' in a general way, rather than to describe performance measures (see Table 3).

The data sets reported: It is not always clearly reported in papers whether the data analysed is from new or evolving systems. The basis of data sets has important implications for the type of metrics that should be used, as change data is highly relevant to evolving system though not relevant to new systems. This lack of information reported in reviewed studies limits the conclusions that we can draw from such studies. Also most studies assume the severity of faults to be equal, yet clearly some faults will be more severe than others in terms of how they affect the running of the system and how long they take to fix.

4. Results

This section presents the results obtained from our analysis in response to our research questions. Before we specifically address each of our research questions, we provide some publication information for the studies we have reviewed.

Date published	Approach used (number of studies)					
	Machine Learning	Stats	Both	Other	Not clear	Total
2000	1	7	3	2	0	13
2001	0	4	0	0	0	4
2002	4	8	2	2	0	16
2003	3	4	0	0	0	7
2004	4	4	5	1	0	14
2005	6	8	3	3	0	20
2006	6	4	8	1	0	19
2007	18	9	5	3	1	36
2008	6	2	3	0	0	11
2009	5	1	1	0	1	8
Total	53	51	30	12	2	148

Table 8. Publications by year

The increase in publications shown in Table 8 indicates a strong interest in studies of fault prediction. This growth seems to have peaked in 2007 and started to tail off thereafter. Machine learning (ML) studies account for much of this growth with the number of studies that use only traditional statistical methods remaining fairly static over the years, but tailing off in the last two years of the review.

Table 9 shows that conference proceedings dominate publications in the area. This is perhaps to be expected since there are many more articles published in conferences than in Journals[5]. Appendix C gives full details of the conferences and Journals in which the papers are published.

Publication	Number	Percentage
Journal	38	26
Conference or workshop	110	74
Total	148	100

Table 9: Publication sources

4.1 The context of fault prediction models (RQ1)

This question allows us to understand the environment for which the prediction model was developed. We examine context primarily in terms of the origin of systems and the programming language that the model has been developed and tested on. This contextual information allows us to consider the applicability and generalisability of models.

Data used	Number	Percentage
1. OSS	32	22
2. NASA	38	26
3. Industrial	64	43
4. Combined	7	5
5. Academic	3	2
6. Not clear	4	3
Total	148	100

NB: Papers making up the data in each table are not shown. Instead each paper has a code assigned for every table in the paper. All codes are in the SRL reference list attached to the individual paper's reference

Table 10. Origin of data used in fault prediction studies

Table 10 indicates that 43% of the data used in this sample comes from closed source/industrial systems with 22% using data from open source systems (OSS); a quarter of studies were based on NASA's industrially-based projects. This shows that a relatively high proportion of data being used in fault prediction studies have industrial origins.

Language used	Number	Percentage
1. C/C++	78	53
2. Java	29	20
3. Other*	14	9
4. Various	9	6
5. Not given	18	12
Total	148	100

*'Other' languages include Assembly, Fortran, Protel and Ada

Table 11. Languages used in fault prediction studies

[5] Though this difference may even out in the next year or two given that research trends can lag behind in Journals given the significantly longer publications lead times they have compared to conferences.

Table 11 shows that the C/C++ language dominates in the studies. Over half of the models are built by analysing C/C++ code and are specifically applicable to such code. Models based on Java code account for 20% of models.

4.2 The variables used in fault prediction models (RQ2)

This question identifies the form that model inputs and outputs take and the range of indicators of fault proneness used in models.

Unit used	Number	Percentage
1. Module**	64	43
2. Class	27	18
3. Component/unit	18	13
4. File	15	10
5. System	4	3
6. Project	3	2
7. Combination	6	4
8. Other*	9	5
9. Not clear	2	1
Total	148	100

*Includes field faults, programming constructs etc.
**Granularity is classified as 'module' if used by authors. We do not re-interpret how the term is applied.

Table 12. Granularity of dependent variables used

Table 12 shows the granularity at which studies report they have modelled fault prediction. There is a range of granularity, with 43% of studies modelling faults in modules, 18% in classes, and a further 10% in files. The module is typically defined in these studies as a function or subroutine in a piece of code (though what the term 'module' actually defines is open to debate). Overall, the range of granularity and the lack of standard terminology used across studies make comparing across studies particularly difficult. Table 13 shows the independent variables used across the studies.

Predictors/independent variables used	Number (n=148)	Percentage of predictors used	Percentage of studies using predictors
1. Previous fault data	35	16	24
2. Change data	38	18	26
3. Static code metrics	115	52	78
4. Dynamic code metrics	4	2	3
5. Miscellaneous*	27	13	18
Total	219	100	

*Includes implementation of design patterns, testing and inspection metrics, qualitative data, and function points
NB. Many studies use more than one independent variable

Table 13. Independent variables used

Table 13 shows that static code metrics (SCM) are the most common basis for fault prediction; 78% of studies use these metrics (52% of all variables used are static code metrics). The use of these metrics is most common where the NASA data has been used as this includes pre-calculated static code data. Table 14 shows this category in more detail. Table 13 also shows that module fault history data is used in 24% of studies (e.g. [[112]]). This indicates a belief that a module previously found to be faulty may continue to be faulty in the future. Dynamic (i.e. run-time) metrics are only used in 3% of studies in our sample (for example [[37]]). This is low given that dynamically executed faults are of particular relevance to users and will directly

impact the reliability and quality of a system. The limited use of dynamic metrics may be due to the difficulty in collecting this type of data but is also likely to be related to the scope of our search strategy[6].

Static code metrics		Number (n=115)	Percentage of SCMs used
1	Size	29	29
2	General	27	27
3	Structure	17	17
4	Complexity	23	23
5	Other	3	3
Total		99	99

NB: Some studies collect more than one type of SCM

Table 14. Static code metrics

Table 14 breaks down the static code metrics used by the 115 studies using them. Table 14 shows that a spread of measures are used, though further analysis also shows that code 1-4 is a very common pattern of use (this mirrors the processed metrics data provided by NASA). Table 14 does show that the most commonly used metric is size (measured in terms of Lines of Code (LOC)). Complexity and structure are also frequently used metrics. Studies in the 'general' category report using static code metrics, source code metrics or product measures without being specific; this category also includes studies that list many metrics in the SCM class. The use of large sets of metrics as independent variables has been made easier by the availability of automated data extraction techniques such as CPP analyzer, PAMPA (e.g. [[7]]) and Code Surfer[7] tools (e.g. [[4]]).

Outputs used	Number	Percentage
1. Categorical	79	53
2. Continuous	40	27
3. Both	28	19
4. Unclear	1	1
Total	148	100

Table 15. Fault dependent variables used

Table 15 shows how faults are reported by our studies. Table 15 shows that there is a bias towards models reporting on categorical faults (i.e. a module is either faulty or non-faulty). However 19% of studies model both variables. As categorical and continuous data often uses different performance measures, we divide the studies in this way when answering the research question relating to model performance.

4.3 The modelling methods used in fault prediction models (RQ3)

This question identifies whether models are based on statistics, machine learning or other approaches. The answer to this question allows us to discuss the popularity and use of particular modelling approaches.

[6] 'Failure' and related reliability-based terms were omitted from our search strategy as we chose to focus on faults in static code.

[7] http://www.grammatech.com/products/codesurfer/

Modelling method used	Number	Percent
1. Machine learning	53	36
2. Statistics	51	35
3. Machine learning & statistics	30	20
5. Other*	12	8
6. Not clear	2	1
Total	148	100

*Includes change propagation; capture-recapture and 'recency' weighting.

Table 16. Modelling method used

		Dependent variable									
		1. Cat (n=79)		2. Cont (n=40)		3. Both (n=28)		4. Unclear (n=1)		Total (n=148)	
		#	%	#	%	#	%	#	%	#	%
Modelling method	1. Machine learning	43	83	3	6	7	13	0	0	53	100
	2. Statistics	15	29	25	49	11	22	0	0	51	100
	3. Machine learning & statistics	16	53	7	23	6	20	1	3	30	100
	5. Other*	3	25	5	42	4	33	0	0	12	100
	6. Not clear	2	100	0	0	0	0	0	0	2	100

*Includes change propagation; capture-recapture and 'recency' weighting.

Table 17. Dependent variable and modelling method used

Table 16 gives a high level view of the methods used to predict faults in the studies. Table 16 appears to show a fairly even balance between machine learning and statistically based studies. However, no less than 30 studies (20%) use both machine learning and statistics. This proportion of studies is slightly deceiving as most of these studies are machine learning studies which also use statistics to differentiate between the performances of several machine learners. This means that the number of machine learning studies could be interpreted as higher than the stated 53 (35%).

Table 17 shows the relationship between the dependent variable and the method used. It shows that 83% of machine learning studies report categorical data with 13% reporting both categorical and continuous data. Studies based on statistical methods report 49% continuous data with the remainder reporting either categorical or both types of data.

Approach used	Number (n=83 studies)	Percentage of approaches used
1. Statistical classifiers	36	24
2. Nearest neighbour methods	12	8
3. Neural networks	24	16
4. Support vector machine-based	8	5
5. Decision tree approaches	28	19
6. Ensemble methods	11	7
7. Misc	31	21
Total	150	100

Table 18. Classification of machine learning approaches

Table 18 shows the type of classification approaches that the 83 studies using machine learning employ (these 83 studies are made up of the 53 that use only machine learning together with the 30 that use both machine learning and statistical methods). Table 18 shows that a spread of classifiers is used. Statistical classifiers are most popular but decision tree approaches and

neural networks are also fairly common. Table 18 also shows that 150 classifiers are used by 83 studies. This shows that many studies use more than one classifier and it is quite common for studies to compare the performance of a variety of learners (e.g. [[4]]).

Statistics used	Number (n=81 studies)	Percentage
1. Regression	46	47
2. Compare means	7	7
3. Correlation	14	15
4. Feature selection	9	9
5. Descriptive stats	5	5
6. Other	16	16
Total	97	100

Table 19. Statistical methods used

Table 19 shows the statistical methods used by 81 studies (50 based only on statistical methods and 31 based on using machine learning and statistical methods). Table 19 shows that by far the most popular approach is regression, almost half of the studies are based on using some form of regression. Table 19 also shows that fewer statistically based studies use multiple methods, with 97 methods being used by 81 studies.

4.4 Measuring the performance of fault prediction models (RQ4)

The studies use a variety of either statistically or machine learning based measures to test the reliability of their results. To analyse these, we divide the studies into whether they report categorical or continuous faults. This is a simplified division as only 25 (49%) statistical studies (Table 17) report only continuous data. Many other studies use both statistical and machine learning methods and report continuous and categorical data and the performance indicators used by these studies is also shown.

Performance Indicator	Continuous studies (n=40)		Studies using both continuous and categorical data (n=28)	
	Number	Percent	Number	Percent
4. Error rates (MRE,MAE etc)	13	33	7	32
5. Regression coefficient	12	30	6	27
6. Correlation test	8	20	5	23
7. Variance significance test	6	15	3	14
9. Other	1	3	1	5
Total (studies may use more than one or none)	40	100	22	100

Table 20. Performance measures used in studies using continuous data

Table 20 shows that a range of tests used to demonstrate the performance of models based on continuous output data. This range corresponds with the range of methods used and the tests necessary for those methods. Table 20 shows that the use of error rates are most popular followed by regression coefficients. These results correspond to the use of regression by 47% of statistical studies as shown in Table 19.

Performance Indicator	Categorical studies (n=79)		Studies using both Categorical & Continuous data (n=28)	
	Number	Percent	Number	Percent
1. Confusion Matrix related composite measures	53	59	16	76
2. Confusion Matrix constructs	32	41	5	24
Total (studies may use more than one or none)	85	100	21	100

Table 21. Performance measures used in categorical studies

Table 21 shows the performance measures used by categorical studies. It shows that many confusion matrix composite measures and constructs are reported by papers. Almost all studies reporting categorical data use at least one of these performance indicators. However, the range of measures that can be used and the combinations selected by authors is large. This makes comparing performance across categorical studies particularly difficult. This problem is compounded with some studies reporting the performance of their model without explicitly describing the performance indicators on which this performance is based (for example [[115]]).

	Categorical studies (n=79)	
Is a confusion matrix provided?	#	%
1. Yes	11	14
2. No	68	86
Total	79	100

Table 22. Provision of a confusion matrix

Comparison of categorical model performance is more practical with the provision of a raw confusion matrix. Table 22 shows that not many studies explicitly provide the raw confusion matrix. Most categorical studies have probably produced a confusion matrix in order to calculate indicators such as *recall* or *balance*, but authors have not reported this. However, with multiple validation techniques being used this may not be possible (a single experiment with stratified 10-fold cross validation repeated 1000 times on one dataset may yield typically 10,000 individual matrices). The provision of this raw data would be very useful for future researchers and potential users of models as it significantly simplifies comparison across studies. Pizzi et al [[133]] provide a very usable format for presenting a confusion matrix.

	Data used					
	Categorical		Both		Total	
Data balancing applied?	#	%	#	%	#	%
0. Data already balanced	2	3	0	0	2	2
1. Yes	9	11	3	11	12	11
2. No*	68	86	25	89	93	87
Total	79	100	28	100	107	100

* We also include studies where it is not clear that balancing has been done

Table 23. Data balancing used in categorical studies

Table 23 shows explicit data balancing has been applied by categorical studies and studies using both categorical and continuous data. Table 23 shows that 12 of 107 of these studies are based on balanced data sets. This suggests that the vast majority of studies either base their

predictions on imbalanced data or else do not report the data balancing that they have applied. Although data balancing is most important for studies reporting categorical data, some studies reporting continuous data also report that they balanced the data (for example [[3]]).

Data balancing applied?	Accuracy or error rate reported #	%
0. Data already balanced	1	3
1. Yes	8	22
2. No*	28	76
Total	37	100

* Included are studies where it is not clear that balancing has been done

Table 24. Use of the accuracy performance measure and data balancing

The importance of balanced data is related to the performance indictors used. Balanced data is particularly important to the credibility of studies which report model performance in terms of accuracy or error rate. Table 24 shows studies explicitly reporting model performance in terms of accuracy and error rate in relation to data balancing. Table 24 shows that only 9 of the 37 studies (25%) reporting model performance in terms of accuracy are based on balanced data sets.

		Cross-validation reported? Yes #	%	No #	%	Total #	%
Methods	1. Machine learning	26	47	27	53	53	100
	2. Statistics	10	20	41	80	51	100
	3. Machine learning & statistics	24	77	6	23	30	100

* Total is not 148 as studies using 'other' methods or where methods are not clearly stated are excluded

Table 25. Validation used in categorical studies

Table 25 shows how often cross-validation is explicitly reported in studies (for example the use of 10-fold cross validation). Cross-validation is most relevant to improving the internal validity of machine learning models, but also can be applied to improve the robustness of statistical models. Table 25 shows that under half of machine learning studies report applying cross-validation. It may be that studies perform cross-validation but do not explicitly report doing so. Very few studies address the external validity of their model, an example that does is [[73]].

5. Discussion

In this section we comment on the implications of what we found in response to our research questions. Overall, the results reported in the papers highlight the many different approaches taken to tackle the problem of producing accurate, reliable and credible fault prediction models. The amount of difference in approaches makes it difficult to compare across studies. This problem is well documented with, for example Runeson et al (2006) noting that "comparing studies is difficult" due to differences in fault types, the few faults found in many studies and subjective classifications of faults.

5.1 Model context (RQ1)

Although the number of papers reporting fault prediction models seems to be decreasing, the sophistication of papers has improved. For example, many more papers are now aware of the

complexity of measuring the performance of models and are addressing model performance more credibly (for example, Arisholm et al [2] is published just beyond our cut off date but is an excellent example of a thoroughly mature approach).

NASA's publicly available software metrics data have proved very popular in developing fault prediction models. And has the advantage that researchers are able to replicate and compare results using different approaches based on the same data set. However, although the repository holds many metrics and is publicly available, it is not possible to explore the source code or trace back how the metrics were extracted. It is also not always possible to identify if any changes have been made to the extraction and computation mechanisms over time. A further concern is that the data may suffer from 'noise' [[44]]. 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. [[62]] point out, NASA works in a unique niche market developing software which is not typical of the generality of software systems, though Turhan et al [[144]] have demonstrated that models built on NASA data are useful for predicting faults in washing machines.

Although it is generally agreed that model performance varies across different software engineering contexts (e.g. [[67]]), there are a few exceptions to this. For example, Bellini et al [[7]] found a metric based on a count of different data structures that was not dependent on the project. Zhou and Leung [[107]] examined relationships between OO design metrics and fault-proneness of classes and agreed with other researchers that the Chidamber and Kemerer [7] Metrics: Weighted Methods for Class (WMC), Response For a Class (RFC), Coupling between Objects (CBO) were almost consistently found to be statistically significant to fault-proneness of classes.

5.2 Model variables (RQ2)

Our results show that researchers are still struggling to find reliable and useful metrics as an input to their fault prediction models. Researchers continue to search for predictor variables that correlate to faults independent of project context. Others suspect that the relationship between software metrics and faults is not strong enough for accurate predictions (e.g. [[48]]). Fenton and Ohlsson [[24]] question the wide use of complexity metrics, and believe that 'there is still no conclusive evidence that such crude metrics are better predictors of fault-proneness than something as simple as LOC.…". Our results indicate that simple LOC are not always a reliable predictor of fault-proneness, for example [[47]], especially when used as the only predictor. Indeed, it is unlikely that any universal predictor for software faults that works for all software projects exists. The metric that performs best will always depend on the context. However we remain some distance from determining which model is most general, i.e., that works in most situations.

Our results show a wide variation in code granularity on which models are based. This makes comparing across studies complex and may make it difficult to repeat studies. Furthermore, a definition of granularity is seldom given in papers. For example, many studies are based on predicting faults in 'modules' though the definition of 'module' is not often given and when it is, varies enormously.

5.3 Modelling approaches (RQ3)

In our sample we found that Machine Learning (ML) methods are increasing. It could be that this increase is due to the methodological problems found in the more traditional statistical

approaches [9], or the increasing availability of easy to use data mining tools such as Weka (http://www.cs.waikato.ac.nz/ml/weka/) and LibSVM (http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/). However, despite the substantial level of research activity and the many different models developed in the area of fault prediction, there is still no consensus in the research community as to which approach is most applicable to specific environmental circumstances ([14] and [1]). Furthermore, the use of such approaches in this context is not yet mature. Many studies focus on comparing the performance of a wide variety of machine learners. While many researchers report in detail how predictive performance is influenced by the various calibrations of machine learners. Menzies et al [[135]] have recently cast doubt on the value of continuing to seek improved fault prediction performance via the increasing calibration of machine learners. They report that a ceiling on the performance of machine learners in this domain has now been reached.

5.4 Measuring model performance (RQ4)

Meta-analysing the performance of fault prediction models is very challenging. In this section we discuss the many factors contributing to this problem (the terms used are described in Table 3 in the Background Section).

The use of multiple measures. Many studies report multiple performance measures produced using multiple methods. This makes any meta-analysis of models a complex issue and impractical across all fault prediction models. This presentation of multiple measures is probably related to the use of tools like Weka which do this automatically. While this may add internal validity to individual studies, it is unclear how to extend these results to improve confidence in their external validity.

Lack of standardisation. The current lack of standardisation in reporting the performance of fault prediction models means that it is often difficult to interpret which performance measures have actually been used and to identify the contextual information. For example, it is not always apparent whether the data has been balanced or whether it is necessary to do so; lacking such information means we cannot be certain of the reliability of measures. In addition, basic information about model performance is often not provided in studies. This makes it practically impossible to compare model performances across studies. In particular the omission of raw confusion matrices makes it problematic to compare the fundamental performances of models. Although finding a way to present potentially multiple confusion matrices is a challenge for researchers. Khoshgoftaar [[147]] has provided a suggested way of doing this.

The selection of performance measures. Selecting appropriate performance measures is complicated and some measures may be better than others at reflecting how well a model performs in a particular context. Indeed, the use of a particular performance measure can be misleading and seemingly inflate or deflate a model's performance. For example, 55% accuracy on a balanced data set may imply a higher level of predictive performance than 99% accuracy on an imbalanced data set, even though the latter sounds more impressive. A more extreme example is a model that simply states that *all* units are non-faulty and can achieve 80% accuracy for a system in which 20% of the units were actually faulty. In these situations even though precision and recall decrease, accuracy might increase (or vice versa).

Categorical studies reporting accuracy or error rate/misclassification rate (=1-accuracy) figures for imbalanced test sets do not tell the reader anything useful about the model's performance. Such papers should be treated with caution - when dealing with data that is imbalanced such measures typically present no information about the ability of the model to detect faults.

Even when the data is balanced and accuracy is reported, it still may not be the most appropriate performance measure to report because these measures do not take into account the cost of mis-classification. This cost can be a significant issue in safety critical software engineering domains. Contextual factors must be considered when selecting appropriate performance measures, for example the cost of a fault escaping discovery. For critical faults, like security faults, the escape cost is high. In this context a manager might want to have high recall but can live with lower precision. For arbitrary faults, a manager might want higher precision. Ultimately cost should be considered in the performance measure. We found few studies consider such issues, those that do include [[3]].

Our results suggest that it is only relatively recently that the complexity of these issues in the selection of performance measures is being addressed with some notable papers showing increasing maturity. Improved understanding of how performance measures behave is also contributing to this growing maturity, for example Menzies et al [48] show that the values of precision vary greatly when validating models across different data sets.

Data balancing.
Our findings show that not all papers mention class distributions and the problem of working with 'imbalanced' datasets have yet to be resolved. The analysis of imbalanced data is frequently performed in the machine learning community as well as the general software engineering community with some datasets having as few as 0.4% defective modules (MDP dataset PC2). However such data should be used with caution. As explained in [[147]], the baseline for comparing performance could be 'does the model do better than predicting the majority class?' This would be a way of determining if the model is telling us something new and would discount the apparent 'excellent' and potentially spurious findings of some papers. However [[139]] argues that it is the minority class (defective) which we are most interested in and a 'predict majority class' model is not particularly useful.

In an ideal world, data mining models should be trained on data sets that have an equal number of each output class (i.e. an equal number of defective and non-defective modules) since this is commonly what the default algorithms expect. Some studies ([[82]], [[118]]) have taken steps to achieve balanced data using techniques such as over-sampling or under-sampling. Unfortunately, these techniques can add problems by reducing the number of tuples in the training set (under-sampling) or create zero existent tuples or duplicate tuples (over-sampling). When the data is balanced, the performance measure of accuracy can be applied with some meaning, however accuracy should not be used in conjunction with imbalanced data as demonstrated by [[147]].

Statistical significance. Not all of the studies include reports of statistical significance tests. Our review supports the previous findings of Lessmann *et al.* [[51]] that "statistical hypothesis testing has only been applied to a very limited extent in the software fault prediction literature. …. it is standard practice to derive conclusions without checking significance". It is not reliable to draw conclusions merely on observed differences because the differences might have been caused by chance alone [1]. This means that it is not possible to be sure of the significance of the results obtained in over half the papers in this study. That said, there are some valuable studies reporting insignificant statistical results (e.g. [[112]], [[24]]). These studies contribute important information since while it is useful to learn about methods that are successful, it is also essential to understand what does not work; this information will save time in future fault prediction studies.

Validation. The credibility of a model's reported performance also depends on whether it has been externally or internally validated. Very few studies apply their model in a variety of settings, and those that do have variable results. Our knowledge of how well these models perform outside their given domain is limited. Our findings support Challagulla et al. 's [[17]] view that there remains a predominant focus on internal validity and a general failure to demonstrate external validity.

6. Conclusions

This review shows that many fault prediction models have been published in the last ten years. These models are heterogeneous and come in all shapes and sizes. A wide range of statistical and machine learning approaches are used to build models. An encouragingly high proportion of studies have been developed using industrial data, with OSS data also featuring strongly. Many studies use the publicly available NASA data. A wide variety of independent variables have been used in models, the most common of which fall into categories such as static code metrics, change metrics and previous fault metrics. However, there is no clear 'best' indicator of fault proneness emerging from studies, with indicators performing differently across different studies. Independent variables such as lines of code, complexity metrics, process metrics, module size, age of file were strongly correlated to faults in some studies but had no correlation to faults in others. This suggests that there is no single 'best' approach to predicting faults across all problem domains. The challenge remains to identify the context variables that determine a model's applicability.

The current state of the literature makes it complex and difficult to evaluate models. This is likely to be a barrier to practitioners implementing these models and makes the meta-analysis of model performance across the whole literature currently infeasible. A number of separate meta-analyses of segments of the literature may be possible. However, even this is a challenge given the wide range of performance indicators used by studies and the inter-dependence of these with other factors in the model.

Currently there is no standardised way of presenting essential model building data. This makes it more difficult to evaluate the performance of a model as essential raw data (for example confusion matrix data) is often not reported. Other data is presented differently across studies (for example accuracy) and some studies do not give the basis of the measures used to measure performance. Although our review suggests that fault prediction modelling is maturing, there remain many methodological anomalies and omissions in the majority of published studies. This makes it difficult for researchers to accurately replicate previous studies or to transparently build on previous studies. Maturity in the area seems to be slower than it could be as not all studies build on the good practices of previous work.

To encourage the uptake of such good practices we are currently preparing comprehensive guidelines for the design and presentation of fault prediction studies in software engineering. These recommendations are based on the findings we report in this paper and should go some way towards helping researchers to provide studies that can be more effectively built on in the future.

Acknowledgements

We are grateful to the UK's Engineering and Physical Science Research Council who supported this research under grant EPSRC EP/E063039/1 and to Professor Martin Shepperd who helped particularly in identifying the different types of faults these studies modelled. We also

acknowledge Dr Paul Wernick who provided input to the early stages of the work for this paper.

Main References

[1] Afzal, W. and R. Torkar (2008). Lessons Learned in Evaluating Software Engineering Prediction Systems. Software Productivity Analysis and Cost Estimation, The 2nd International Workshop, SPACE 2008. 2 December, Beijing, China 35-43
[2] Arisholm E, Briand L, Johannessen E: A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. Journal of Systems and Software 83(1): 2-17 (2010)
[3] Batista, G. E., Prati, R. C., and Monard, M. C. 2004. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explor. Newsl.* 6, 1 (Jun. 2004), 20-29
[4] Beecham S, Hall T, Bowes D, Gray D, Counsell S, Wernick P, Black S, Leibchen G (2009) A short comparison of two systematic literature reviews on fault prediction, University of Hertfordshire Tech Rep UH-CS-TR-492
[5] Catal, C. and B. Diri (2009). "A systematic review of software fault prediction studies." Expert Systems with Applications 36(4): 7346-7354
[6] Chawla, N. V., K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer (2002). "SMOTE: Synthetic Minority Over-sampling Technique." Journal of Artificial Intelligence Research 16: 321–357.
[7] Chidamber, S. R. and C. F. Kemerer (1994). "A metrics suite for object oriented design." IEEE Transaction on Software Engineering 20: 476-493.
[8] Dieste O, Grimán A, Juristo N (2009) "Developing search strategies for detecting relevant experiments", Empirical Software Engineering Journal, 14 (5), 513- 539
[9] Fenton, N. E. and M. Neil (1999). "A critique of software defect prediction models." Software Engineering, IEEE Transactions on 25(5): 675-689..
[10] Jørgensen, M. and M. Shepperd (2007). "A Systematic Review of Software Development Cost Estimation Studies." IEEE Transactions on Software Engineering 33(1): 33 - 53
[11] Kamei, Y.; Monden, A.; Matsumoto, S.; Kakimoto, T.; Matsumoto, K. (2007) The Effects of Over and Under Sampling on Fault-prone Module Detection, Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on
[12] Kitchenham, B. (2004). Procedures for Performing Systematic Reviews, Keele University and National ICT Australia Ltd.: 1 - 28.
[13] MacDonell, S. G. and M. J. Shepperd (2007). Comparing Local and Global Software Effort Estimation Models -- Reflections on a Systematic Review. Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on 401-409.
[14] Myrtveit, I., E. Stensrud and M. Shepperd (2005). "Reliability and validity in comparative studies of software prediction models." IEEE Transactions on Software Engineering 31(5).
[15] Ostrand, T. J. and E. J. Weyuker (2007). How to measure success of fault prediction models. Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting, Dubrovnik, CroatiaACM 25-30.
[16] Runeson, P., C. Andersson, T. Thelin, A. Andrews and T. Berling (2006). "What do we know about defect detection methods? [software testing]." Software, IEEE 23(3): 82-90.
[17] Runeson, P., M. C. Ohlsson and C. Wohlin (2001). A Classification Scheme for Studies on Fault-Prone Components. PROFES 2001 341-355.
[18] Sun, Y., M. Robinson, R. Adams, R. t. Boeckhorst, A. G. Rust and N. Davey (2006). Using sampling methods to improve binding site predictions (Neural Networks and Machine Learning in Bioinformatics - Theory and Applications). Artificial Neural Networks (ESANN2006), 14th Euro Symp on, Bruges, Belgium, April 26-28.

References for the 148 included SLR papers [[1-148]]

(Note that references from this list are cited using the format [[ref#]])

Each reference is followed by the categorisations suggested by the authors of this paper. The codes relate to the data presented in the data tables (Tables 10-25) within the paper. For example: T10=1 means that for the purposes of Table 10 (Data used) a paper was coded '1' (i.e. using OSS data).

Please report possible misclassifications and missing papers to: tracy.hall@brunel.ac.uk

[[1]] Amasaki, S., Y. Takagi, O. Mizuno, and T.A.K.T. Kikuno (2003). A Bayesian belief network for assessing the likelihood of fault content in Software Reliability Engineering, ISSRE, International Symposium. p. 215-226.

(Paper=1; T10=3; T11=5; T12=6: T13=1,3,5; T14=1; T15=1; T16=1; T18=1; T19=NA; T20/21=1,7,4; T22=2; T23=2; T24=y; T25=n)

[[2]] Andersson, C. and P. Runeson (2007). A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems. Software Engineering, IEEE Transactions on. 33(5): p. 273-286.
(Paper=2; T10=3; T11=1; T12=1: T13=2,3; T14=2; T15=2; T16=2; T18=NA; T19=3, 6; T20/21=6; T22=2; T23=2; T24=n; T25=n)

[[3]] Arisholm, E. and L.C. Briand (2006). Predicting fault-prone components in a java legacy system in Empirical Software Engineering, Procs of the 2006 ACM/IEEE International Symposium on. Rio de Janeiro. p8-17.
(Paper=3; T10=3; T11=2; T12=2: T13=2,3; T14=2; T15=3; T16=2; T18=NA; T19=1, 4, 6; T20/21=2; T22=2; T23=1; T24=n; T25=y)

[[4]] Arisholm, E., L.C. Briand, and M. Fuglerud (2007). Data Mining Techniques for Building Fault-proneness Models in Telecom Java Software in Software Reliability, 2007. ISSRE '07. The 18th IEEE International Symposium on. p. 215-224.
(Paper=4; T10=3; T11=2; T12=2: T13=1,2,3,5; T14=5; T15=1; T16=1; T18=3 4 5 7; T19=NA; T20/21=1,2; T22=2; T23=1; T24=n; T25=n)

[[5]] Ayewah, N., W. Pugh, J.D. Morgenthaler, J. Penix, and Y. Zhou (2007). Evaluating static analysis defect warnings on production software in Workshop on Program analysis for software tools and engineering, Proceedings of the 7th ACM SIGPLAN-SIGSOFT San Diego, California, USA. p. 1-8: ACM.
(Paper=5; T10=4; T11=2; T12=3: T13=1,5; T14=NA; T15=3; T16=5; T18=NA; T19=NA; T20/21=2; T22=2; T23=2; T24=n; T25=n)

[[6]] Bell, R.M., T. Ostrand, J. , and E.J. Weyuker (2006). Looking for bugs in all the right places in Software testing and analysis, Proceedings of the 2006 International symposium on Portland, USA. p1-72: ACM.
(Paper=6; T10=3; T11=2; T12=4: T13=1,2,3; T14=1; T15=2; T16=2; T18=NA; T19=1.1; T20/21=5; T22=2; T23=2; T24=; T25=n)

[[7]] Bellini, P., I. Bruno, P. Nesi, and D. Rogai (2005). Comparing fault-proneness estimation models in Engineering of Complex Computer Systems (ICECCS'05), Proc. of 10th IEEE International Conference on p. 205–214.
(Paper=7; T10=3; T11=1; T12=3: T13=3; T14=2; T15=1; T16=2; T18=NA; T19=1, 2, 3; T20/21=1,2; T22=2; T23=2; T24=n; T25=y)

[[8]] Bernstein, A., J. Ekanayake, and M. Pinzger (2007). 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. Dubrovnik, Croatia. p. 11-18: ACM.
(Paper=8; T10=1; T11=2; T12=7: T13=2; T14=NA; T15=3; T16=1; T18=1 5 7; T19=NA; T20/21=1,6,7; T22=1; T23=2; T24=y; T25=n)

[[9]] Bezerra, M.E.R., A.L.I. Oliveira, and S.R.L. Meira (2007). A Constructive RBF Neural Network for Estimating the Probability of Defects in Software Modules in Neural Networks, 2007. IJCNN 2007. International Joint Conference on. p. 2869-2874.
(Paper=9; T10=2; T11=1; T12=1: T13=1,3; T14=1; T15=1; T16=1; T18=1 3; T19=NA; T20/21=1; T22=2; T23=2; T24=y; T25=y)

[[10]] Bibi, S., G. Tsoumakas, I. Stamelos, and I. Vlahvas (2006). Software Defect Prediction Using Regression via Classification in Computer Systems and Applications, 2006. IEEE International Conference on. p. 330-336.
(Paper=10; T10=3; T11=3; T12=2: T13=5; T14=NA; T15=2; T16=3; T18=6; T19=Not clear; T20/21=4; T22=2; T23=2; T24=y; T25=y)

[[11]] Binkley, D., H. Feild, D. Lawrie, and M. Pighin (2007). Software Fault Prediction using Language Processing in Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007. p. 99-110.
(Paper=11; T10=1; T11=1; T12=1: T13=2,3; T14=1; T15=2; T16=2; T18=NA; T19=1; T20/21=5; T22=2; T23=2; T24=n; T25=n)

[[12]] Briand, L.C., K. El Emam, B.G. Freimut, and O. Laitenberger (2000). A comprehensive evaluation of capture-recapture models for estimating software defect content. Software Engineering, IEEE Transactions on. 26(6): p. 518-540.
(Paper=12; T10=4; T11=1; T12=8: T13=5; T14=NA; T15=2; T16=5; T18=NA; T19=NA; T20/21=4; T22=2; T23=2; T24=n; T25=y)

[[13]] Briand, L.C., W.L. Melo, and J. Wu (2002). Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects. Software Engineering, IEEE Transactions on. 28(7): p. 706-720.
(Paper=13; T10=3; T11=2; T12=2: T13=3; T14=1,4,5; T15=3; T16=2; T18=NA; T19=1.2, 1.3 4, 6; T20/21=1; T22=2; T23=2; T24=n; T25=y)

[[14]] Catal, C., B. Diri, and B. Ozumut (2007). An Artificial Immune System Approach for Fault Prediction in Object-Oriented Software in Dependability of Computer Systems, 2007. DepCoS-RELCOMEX '07. 2nd International Conference on. p. 238-245.
(Paper=14; T10=2; T11=1; T12=2: T13=3; T14=1,5; T15=1; T16=1; T18=7; T19=NA; T20/21=1; T22=1; T23=2; T24=n; T25=y)

[[15]] Ceylan, E., F.O. Kutlubay, and A.B. Bener (2006). Software Defect Identification Using Machine Learning Techniques in Software Engineering and Advanced Applications, 2006. SEAA '06. 32nd EUROMICRO Conference on. p. 240-247.
(Paper=15; T10=3; T11=5; T12=1: T13=1,3; T14=2; T15=2; T16=3; T18=3 5 7; T19=Not clear; T20/21=3,4; T22=2; T23=2; T24=n; T25=n)

[[16]] Challagulla, V.U.B., F.B. Bastani, and I.L. Yen (2006). A Unified Framework for Defect Data Analysis Using the MBR Technique in International Conference on Tools with Artificial Intelligence, 2006. ICTAI '06. 18th IEEE p. 39-46.
(Paper=16; T10=2; T11=1; T12=1: T13=3; T14=1,4; T15=1; T16=1; T18=2 5 6 7; T19=NA; T20/21=1,2; T22=2; T23=2; T24=y; T25=n)

[[17]] Challagulla, V.U.B., F.B. Bastani, I.L. Yen, and R.A. Paul (2005). Empirical assessment of machine learning based software defect prediction techniques in Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005. 10th IEEE International Workshop on. p. 263-270.
(Paper=17; T10=2; T11=1; T12=1: T13=3; T14=1,5; T15=2; T16=3; T18=1 2 3 5 6 7; T19=Not clear; T20/21=4; T22=2; T23=2; T24=n; T25=n)

[[18]] Dallmeier, V. and T. Zimmermann (2007). Extraction of bug localization benchmarks from history in Automated Software Engineering, Proceedings of the twenty-second IEEE/ACM International conference on Atlanta, Georgia, USA. p. 433-436: ACM.
(Paper=18; T10=1; T11=2; T12=3: T13=3 1; T14=2; T15=3; T16=5; T18=NA; T19=NA; T20/21=6,1; T22=2; T23=2; T24=n; T25=n)

[[19]] Denaro, G. (2000). Estimating software fault-proneness for tuning testing activities in Software Engineering, Proceedings of the 22nd International Conference on Limerick, Ireland. p. 704-706: ACM.
(Paper=19; T10=3; T11=1; T12=1: T13=3; T14=1,2,4; T15=1; T16=2; T18=NA; T19=1.2; T20/21=5,7; T22=2; T23=2; T24=n; T25=n)

[[20]] Denaro, G., S. Morasca, and M. Pezz (2002). Deriving models of software fault-proneness in Software engineering and knowledge engineering, Proceedings of the 14th International conference on Ischia, Italy. p. 361-368: ACM.
(Paper=20; T10=3; T11=1; T12=1: T13=3,4; T14=2; T15=1; T16=2; T18=NA; T19=1.2; T20/21=1; T22=2; T23=2; T24=n; T25=y)

[[21]] Denaro, G. and M. Pezz (2002). An empirical evaluation of fault-proneness models in Software Engineering, Proceedings of the 24th International Conference on Orlando, Florida. p. 241-251: ACM.
(Paper=21; T10=1; T11=1; T12=1: T13=3; T14=1,2,4,7; T15=1; T16=2; T18=NA; T19=1, 4; T20/21=1,5; T22=2; T23=2; T24=n; T25=y)

[[22]] El Emam, K., W.L. Melo, and J.C. Machado (2001). The prediction of faulty classes using object-oriented design metrics. Journal of Systems and Software. 56(1): p. 63-75.
(Paper=22; T10=3; T11=2; T12=2: T13=3; T14=2; T15=1; T16=2; T18=NA; T19=1, 3, 5; T20/21=5,2,1,7,4; T22=1; T23=2; T24=n; T25=n)

[[23]] Fenton, N., M. Neil, W. Marsh, P. Hearty, L. Radlinski, and K. Paul (2007). Project Data Incorporating Qualitative Facts for Improved Software Defect Prediction in Predictor Models in Software Engineering, Proceedings of the Third International Workshop on IEEE Computer Society.
(Paper=23; T10=3; T11=5; T12=6: T13=1,3,5; T14=1; T15=2; T16=1; T18=1; T19=NA; T20/21=5; T22=2; T23=2; T24=n; T25=n)

[[24]] Fenton, N.E. and N. Ohlsson (2000). Quantitative analysis of faults and failures in a complex software system. Software Engineering, IEEE Transactions on. 26(8): p. 797-814.
(Paper=24; T10=3; T11=1; T12=1: T13=1,2,3; T14=2; T15=2; T16=2; T18=NA; T19=3, 6; T20/21=6,2; T22=2; T23=2; T24=n; T25=n)

[[25]] Fioravanti, F. and P. Nesi (2001). A Study on Fault-Proneness Detection of Object-Oriented Systems in European Conference Software Maintenance and Reeng. (CSMR 2001), Proc. Fifth p. 121-130.
(Paper=25; T10=6; T11=1; T12=5: T13=3; T14=5; T15=1; T16=2; T18=NA; T19=1.2, 4; T20/21=1; T22=2; T23=1; T24=y; T25=n)

[[26]] Gao, K. and T.M. Khoshgoftaar (2007). A Comprehensive Empirical Study of Count Models for Software Fault Prediction. Reliability, IEEE Transactions on. 56(2): p. 223-236.
(Paper=26; T10=3; T11=1; T12=1: T13=2,3,5; T14=1; T15=2; T16=2; T18=NA; T19=1.2; T20/21=4; T22=2; T23=2; T24=n; T25=y)

[[27]] Graves, T.L., A.F. Karr, J.S. Marron, and H.A.S.H. Siy (2000). Predicting fault incidence using software change history. Software Engineering, IEEE Transactions on. 26(7): p. 653-661.
(Paper=27; T10=3; T11=1; T12=3: T13=2,3; T14=1,4; T15=2; T16=2; T18=NA; T19=1.3; T20/21=5,4; T22=2; T23=2; T24=n; T25=n)

[[28]] Guo, L., B. Cukic, and H. Singh (2003). Predicting fault prone modules by the Dempster-Shafer belief networks in Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on p. 249-252.
(Paper=28; T10=2; T11=1; T12=1: T13=1,3; T14=1,4; T15=1; T16=1; T18=1 7; T19=NA; T20/21=1; T22=2; T23=2; T24=y; T25=y)

[[29]] Guo, L., Y. Ma, B. Cukic, and S. Harshinder (2004). Robust prediction of fault-proneness by random forests in Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on. p. 417-428.
(Paper=29; T10=2; T11=1; T12=1: T13=3; T14=1,4; T15=1; T16=1; T18=5 7; T19=NA; T20/21=1; T22=2; T23=2; T24=y; T25=y)

[[30]] Gyimothy, T., R. Ferenc, and I. Siket (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. Software Engineering, IEEE Transactions on. 31(10): p. 897-910.

(Paper=30; T10=1; T11=1; T12=2: T13=3; T14=1,5; T15=3; T16=3; T18=1 3 5; T19=Not clear; T20/21=1; T22=2; T23=2; T24=n; T25=y)

[[31]] Harel, A. and K. Kantorowitz (2005). Estimating the number of faults remaining in software code documents inspected with iterative code reviews in Software - Science, Technology and Engineering, 2005. Proceedings. IEEE International Conference on. p. 151-160.
(Paper=31; T10=3; T11=1; T12=1: T13=3,5; T14=1; T15=2; T16=5; T18=NA; T19=NA; T20/21=7; T22=2; T23=2; T24=n; T25=n)

[[32]] Hassan, A.E. and R.C. Holt (2004). Predicting change propagation in software systems in Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on. p. 284-293.
(Paper=32; T10=1; T11=1; T12=4: T13=2; T14=NA; T15=1; T16=1; T18=8; T19=NA; T20/21=1; T22=2; T23=2; T24=n; T25=n)

[[33]] Hassan, A.E. and R.C. Holt (2005). The top ten list: dynamic fault prediction in Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on. p. 263-272.
(Paper=33; T10=1; T11=1; T12=3: T13=1,2; T14=NA; T15=2; T16=1; T18=7; T19=NA; T20/21=3; T22=2; T23=2; T24=n; T25=n)

[[34]] Hovemeyer, D. and W. Pugh (2004). Finding bugs is easy in Object-oriented programming systems, languages and applications, Companion to the 19th annual ACM SIGPLAN conference on Vancouver, BC, CANADA. p. 132-136: ACM.
(Paper=34; T10=1; T11=2; T12=3: T13=1; T14=NA; T15=1; T16=5; T18=NA; T19=NA; T20/21=2; T22=2; T23=2; T24=n; T25=n)

[[35]] Jiang, L., Z. Su, and E. Chiu (2007). Context-based detection of clone-related bugs in Foundations of Software Engineering, Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT International Symposium on Dubrovnik, Croatia. p. 55-64: ACM.
(Paper=35; T10=1; T11=4; T12=3: T13=3; T14=5; T15=3; T16=1; T18=7; T19=NA; T20/21=2; T22=2; T23=2; T24=n; T25=n)

[[36]] Jiang, Y., B. Cukic, and T. Menzies (2007). Fault Prediction using Early Lifecycle Data in Software Reliability, 2007. ISSRE '07. The 18th IEEE International Symposium on. p. 237-246.
(Paper=36; T10=2; T11=1; T12=1: T13=1,5; T14=2; T15=1; T16=3; T18=1 2 3 5 7; T19=Not clear; T20/21=1,2; T22=2; T23=2; T24=n; T25=y)

[[37]] Jones, J.A. and M.J. Harrold (2005). Empirical evaluation of the tarantula automatic fault-localization technique in Automated Software Engineering, Proceedings of the 20th IEEE/ACM International Conference on Long Beach, CA, USA. p. 273-282: ACM.
(Paper=37; T10=3; T11=1; T12=3: T13=4; T14=NA; T15=3; T16=5; T18=NA; T19=NA; T20/21=1,9; T22=2; T23=2; T24=n; T25=n)

[[38]] Jones, J.A., M.J. Harrold, and J. Stasko (2002). Visualization of test information to assist fault localization in Software Engineering, Proceedings of the 24th International Conference on Orlando, Florida. p. 467-477: ACM.
(Paper=38; T10=3; T11=1; T12=3: T13=5; T14=NA; T15=1; T16=5; T18=NA; T19=NA; T20/21=2; T22=2; T23=2; T24=n; T25=n)

[[39]] Joshi, H., C. Zhang, S. Ramaswamy, and C. Bayrak (2007). Local and Global Recency Weighting Approach to Bug Prediction in Mining Software Repositories, 2007. ICSE Workshops MSR '07. Fourth International Workshop on p. 33-33.
(Paper=39; T10=1; T11=2; T12=3: T13=1; T14=NA; T15=2; T16=5; T18=NA; T19=NA; T20/21=8; T22=2; T23=2; T24=n; T25=n)

[[40]] Kaminsky, K. and G. Boetticher (2004). 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. p. 10-15 Vol.1.
(Paper=40; T10=2; T11=1; T12=1: T13=3; T14=2; T15=1; T16=1; T18=7; T19=NA; T20/21=1; T22=2; T23=2; T24=y; T25=n)

[[41]] Kanmani, S., V.R. Uthariaraj, V. Sankaranaryanan, and P. Thambidurai (2004). Object oriented software quality prediction using general regression neural networks. ACM Sigsoft Software Eng. Notes. 29 (5): p.1-6.
(Paper=41; T10=5; T11=1; T12=2: T13=3; T14=5; T15=2; T16=3; T18=1 3; T19=Not clear; T20/21=5,6,3; T22=2; T23=2; T24=n; T25=n)

[[42]] Khoshgoftaar, T.M., K. Gao, and R.M. Szabo (2001). An application of zero-inflated Poisson regression for software fault prediction in Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on. p. 66-73.
(Paper=42; T10=3; T11=1; T12=4: T13=3,5; T14=1; T15=2; T16=2; T18=NA; T19=1.2; T20/21=3; T22=2; T23=2; T24=n; T25=n)

[[43]] Khoshgoftaar, T.M. and N. Seliya (2002). Tree-based software quality estimation models for fault prediction in Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on. p. 203-214.
(Paper=43; T10=3; T11=1; T12=1: T13=3; T14=4,5; T15=3; T16=3; T18=5 6 7; T19=Not clear; T20/21=7,4; T22=2; T23=2; T24=n; T25=y)

[[44]] Khoshgoftaar, T.M. and N. Seliya (2004). Comparative assessment of software quality classification techniques: An empirical study. Empirical Software Engineering. 9(3): p. 229–257.
(Paper=44; T10=3; T11=3; T12=2: T13=3; T14=2; T15=1; T16=3; T18=5 6; T19=Not clear; T20/21=2; T22=2; T23=2; T24=n; T25=y)

[[45]] Khoshgoftaar, T.M., V. Thaker, and E.B. Allen (2000). Modeling fault-prone modules of subsystems in Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium. p259-267.
(Paper=45; T10=3; T11=5; T12=1: T13=2,3,5; T14=2; T15=1; T16=3; T18=5 6; T19=Not clear; T20/21=2; T22=2; T23=2; T24=n; T25=y)

[[46]] Sunghun, K., T. Zimmermann, K. Pan, and E.J.J. Whitehead (2006). Automatic Identification of Bug-Introducing Changes in Automated Software Engineering, 2006. ASE '06. 21st IEEE/ACM International Conference on p. 81-90.
(Paper=46; T10=1; T11=2; T12=6: T13=2; T14=NA; T15=1; T16=1; T18=7; T19=NA; T20/21=2; T22=2; T23=2; T24=n; T25=n)

[[47]] Knab, P., M. Pinzger, and A. Bernstein (2006). Predicting defect densities in source code files with decision tree learners in Mining software repositories, Proceedings of the 2006 International workshop on Shanghai, China. p. 119-125: ACM.
(Paper=47; T10=1; T11=1; T12=4: T13=1,2,3; T14=1,2; T15=3; T16=3; T18=1 5 7; T19=Not clear; T20/21=1; T22=1; T23=2; T24=n; T25=y)

[[48]] Koru, A.G. and H. Liu (2005). Building Defect Prediction Models in Practice. IEEE Software. 22(6): p23-29.
(Paper=48; T10=2; T11=1; T12=1: T13=1,3; T14=1,4,5; T15=3; T16=1; T18=2 7; T19=NA; T20/21=1; T22=2; T23=2; T24=n; T25=y)

[[49]] Koru, A.G., D. Zhang, and H. Liu (2007). Modeling the Effect of Size on Defect Proneness for Open-Source Software in Software Engineering, Companion to the proceedings of the 29th International Conference on p. 115-124: IEEE Computer Society.
(Paper=49; T10=1; T11=1; T12=2: T13=2; T14=NA; T15=3; T16=2; T18=NA; T19=1, 6; T20/21=5,4; T22=2; T23=2; T24=n; T25=n)

[[50]] Kutlubay, O., B. Turhan, and A.B. Bener (2007). A Two-Step Model for Defect Density Estimation in Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on. p. 322-332.
(Paper=50; T10=2; T11=1; T12=1: T13=3; T14=2; T15=3; T16=3; T18=1 5; T19=Not clear; T20/21=1,4; T22=1; T23=2; T24=y; T25=y)

[[51]] Lessmann, S., B. Baesens, C. Mues, and S. Pietsch (2008). Benchmarking classification models for software defect prediction: A proposed framework and novel findings. Software Engineering, IEEE Transactions on. 34(4): p. 485-496.
(Paper=51; T10=2; T11=1; T12=1: T13=1,3; T14=1,4,5; T15=1; T16=3; T18=1 6; T19=Not clear; T20/21=1; T22=2; T23=2; T24=n; T25=y)

[[52]] Li, P.L., J. Herbsleb, and M. Shaw (2005). Finding predictors of field defects for open source software systems in commonly available data sources: a case study of OpenBSD in Software Metrics, 2005. 11th IEEE International Symposium. p. 10 pp.
(Paper=52; T10=1; T11=1; T12=8: T13=2,3,5; T14=2; T15=2; T16=2; T18=NA; T19=1, 3; T20/21=6,3; T22=2; T23=2; T24=n; T25=n)

[[53]] Li, P.L., J. Herbsleb, and M. Shaw (2005). Forecasting field defect rates using a combined time-based and metrics-based approach: a case study of OpenBSD in Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on. p. 10 pp.
(Paper=53; T10=1; T11=1; T12=8: T13=1,2,3,5; T14=1; T15=2; T16=2; T18=NA; T19=1,6; T20/21=9; T22=2; T23=2; T24=n; T25=n)

[[54]] Li, P.L., J. Herbsleb, M. Shaw, and B. Robinson (2006). Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc in Software engineering, Proceedings of the 28th International conference on Shanghai, China. p. 413-422: ACM.
(Paper=54; T10=3; T11=1; T12=3: T13=1,2,3,5; T14=2; T15=2; T16=3; T18=1 3 5; T19=Not clear; T20/21=4; T22=2; T23=2; T24=n; T25=y)

[[55]] Li, P.L., M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam (2004). Empirical evaluation of defect projection models for widely-deployed production software systems in Foundations of Software Engineering, Proceedings of the 12th ACM SIGSOFT twelfth International Symposium on Newport Beach, CA, USA. p. 263-272: ACM.
(Paper=55; T10=4; T11=5; T12=5: T13=5; T14=NA; T15=2; T16=2; T18=NA; T19=1, 4; T20/21=5; T22=2; T23=2; T24=n; T25=n)

[[56]] Li, Z. and M. Reformat (2007). A practical method for the software fault-prediction in Information reuse and integration, IEEE international conference on Las Vegas, Nevada, USA. p. 659–666.
(Paper=56; T10=2; T11=1; T12=8: T13=1,2,3; T14=2; T15=3; T16=1; T18=1 4 7; T19=NA; T20/21=1; T22=2; T23=1; T24=y; T25=y)

[[57]] Liblit, B., A. Aiken, A.X. Zheng, and M.I. Jordan (2003). Bug isolation via remote program sampling in Programming language design and implementation, Proceedings of the ACM SIGPLAN 2003 Conference on San Diego, California, USA. p. 141-154: ACM.
(Paper=57; T10=1; T11=1; T12=3: T13=4; T14=NA; T15=1; T16=2; T18=NA; T19=1.2; T20/21=9; T22=2; T23=2; T24=n; T25=y)

[[58]] Ma, Y., L. Guo, and B. Cukic (2006). A Statistical Framework for the Prediction of Fault-Proneness. Advances in Machine Learning Application in Software Engineering, Idea Group Inc: p. 1-26.
(Paper=58; T10=2; T11=1; T12=1: T13=3; T14=1,4; T15=1; T16=1; T18=5 6 7; T19=NA; T20/21=1,2; T22=2; T23=1; T24=y; T25=y)

[[59]] Madhavan, J.T. and E.J. Whitehead Jr (2007). Predicting buggy changes inside an integrated development environment in OOPSLA workshop on eclipse technology eXchange, Proceedings of the 2007. Montreal, Quebec, Canada. p. 36-40: ACM.
(Paper=59; T10=1; T11=2; T12=4: T13=2,3; T14=4; T15=1; T16=1; T18=4; T19=NA; T20/21=1; T22=2; T23=2; T24=y; T25=y)

[[60]] Malaiya, Y.K. and J. Denton (2000). Module size distribution and defect density in Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium on. p. 62-71.
(Paper=60; T10=4; T11=3; T12=1: T13=1,3; T14=1,4; T15=3; T16=2; T18=NA; T19=3; T20/21=1; T22=2; T23=2; T24=n; T25=n)

[[61]] Marcus, A., D. Poshyvanyk, and R. Ferenc (2008). Using the Conceptual Cohesion of Classes for Fault Prediction in Object-Oriented Systems. Software Engineering, IEEE Transactions on. 34(2): p. 287-300.
(Paper=61; T10=1; T11=1; T12=2: T13=2,3; T14=5; T15=1; T16=2; T18=NA; T19=1, 6, 5; T20/21=1,5; T22=2; T23=2; T24=n; T25=n)

[[62]] Menzies, T., J. Greenwald, and A. Frank (2007). Data Mining Static Code Attributes to Learn Defect Predictors. Software Engineering, IEEE Transactions on. 33(1): p. 2-13.
(Paper=62; T10=2; T11=4; T12=2: T13=3; T14=2; T15=1; T16=1; T18=1; T19=NA; T20/21=1; T22=2; T23=2; T24=n; T25=y)

[[63]] Mizuno, O., S. Ikami, S. Nakaichi, and T. Kikuno (2007). Spam Filter Based Approach for Finding Fault-Prone Software Modules in Mining Software Repositories, Proceedings of the Fourth International Workshop on IEEE Computer Society.
(Paper=63; T10=1; T11=2; T12=1: T13=5; T14=NA; T15=1; T16=1; T18=1 7; T19=NA; T20/21=1; T22=2; T23=1; T24=y; T25=y)

[[64]] Mizuno, O. and T. Kikuno (2007). Training on errors experiment to detect fault-prone software modules by spam filter in Foundations of Software Engineering, Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT International Symposium on. Dubrovnik, Croatia. p. 405-414: ACM.
(Paper=64; T10=1; T11=2; T12=1: T13=5; T14=NA; T15=1; T16=1; T18=1 7; T19=NA; T20/21=1; T22=1; T23=2; T24=y; T25=y)

[[65]] Nagappan, N. and T. Ball (2005). Use of relative code churn measures to predict system defect density in Software engineering, Proceedings of the 27th International conference on. St. Louis, USA. p284-292: ACM.
(Paper=65; T10=3; T11=1; T12=4: T13=2; T14=NA; T15=3; T16=2; T18=NA; T19=1, 2, 3, 4, 6; T20/21=5,4; T22=2; T23=2; T24=n; T25=n)

[[66]] Nagappan, N. and T. Ball (2005). Static analysis tools as early indicators of pre-release defect density in Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on p. 580-586.
(Paper=66; T10=3; T11=1; T12=3: T13=1; T14=NA; T15=2; T16=2; T18=NA; T19=1, 2, 3, 6; T20/21=5,6,2; T22=2; T23=2; T24=n; T25=n)

[[67]] Nagappan, N., T. Ball, and A. Zeller (2006). Mining metrics to predict component failures in Software engineering, Proceedings of the 28th International Conference on Shanghai, China. p. 452-461: ACM.
(Paper=67; T10=3; T11=1; T12=1: T13=1,3; T14=2,5; T15=1; T16=3; T18=1; T19=Not clear; T20/21=5; T22=2; T23=2; T24=n; T25=n)

[[68]] Neufelder, A.M. (2000). How to measure the impact of specific development practices on fielded defect density in Software Reliability Engineering. ISSRE 2000. Procs. 11th International Symposium p148-160.
(Paper=68; T10=3; T11=1; T12=3: T13=5; T14=NA; T15=2; T16=5; T18=NA; T19=NA; T20/21=5,6; T22=2; T23=2; T24=n; T25=n)

[[69]] Nikora, A.P. and J.C. Munson (2003). Developing fault predictors for evolving software systems in Software Metrics Symposium, 2003. Proceedings. Ninth International. p. 338-350.
(Paper=69; T10=2; T11=1; T12=1: T13=2,3; T14=1,5; T15=2; T16=2; T18=NA; T19=1.3; T20/21=5,4; T22=2; T23=2; T24=n; T25=n)

[[70]] Nikora, A.P. and J.C. Munson (2004). The effects of fault counting methods on fault model quality in Computer Software and Applications Conference,. COMPSAC 2004. Proceedings. p. 192-201.
(Paper=70; T10=2; T11=1; T12=1: T13=2,3; T14=2,5; T15=2; T16=2; T18=NA; T19=1, 4; T20/21=5; T22=2; T23=2; T24=n; T25=n)

[[71]] Olague, H.M., S. Gholston, and S. Quattlebaum (2007). 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. 33(6): p. 402-419.
(Paper=71; T10=1; T11=2; T12=2: T13=3; T14=5; T15=1; T16=2; T18=NA; T19=1, 2, 5; T20/21=5,6; T22=2; T23=2; T24=n; T25=n)

[[72]] Oral, A.D. and A.B. Bener (2007). Defect prediction for embedded software in Computer and Information Sciences, 2007. ISCIS 2007. IEEE 22nd International Symposium on. p. 1-6.
(Paper=72; T10=2; T11=1; T12=1: T13=3; T14=2; T15=1; T16=3; T18=1 3 7; T19=Not clear; T20/21=1,2; T22=1; T23=2; T24=n; T25=y)

[[73]] Ostrand, T., J. , E.J. Weyuker, and R.M. Bell (2004). Where the bugs are in Software testing and analysis, Proceedings of the 2004 ACM SIGSOFT International symposium on Boston, Massachusetts, USA. p. 86-96: ACM.

(Paper=73; T10=3; T11=2; T12=4: T13=1,2,3; T14=1; T15=2; T16=2; T18=NA; T19=1.1; T20/21=3; T22=2; T23=2; T24=n; T25=n)

[[74]] Ostrand, T.J. and E.J. Weyuker (2002). The distribution of faults in a large industrial software system in Software testing and analysis, Proceedings of the 2002 ACM SIGSOFT International symposium on Roma, Italy. p. 55-64: ACM.
(Paper=74; T10=3; T11=2; T12=7: T13=1, 3; T14=1; T15=2; T16=2; T18=NA; T19=6, 3; T20/21=3; T22=2; T23=2; T24=n; T25=n)

[[75]] Ostrand, T.J., E.J. Weyuker, and R.M. Bell (2005). Predicting the location and number of faults in large software systems. Software Engineering, IEEE Transactions on. 31(4): p. 340-355.
(Paper=75; T10=3; T11=2; T12=4: T13=2,3; T14=1; T15=3; T16=2; T18=NA; T19=1.1; T20/21=5,1,4; T22=2; T23=2; T24=n; T25=n)

[[76]] Ostrand, T.J., E.J. Weyuker, and R.M. Bell (2005). Locating where faults will be in Diversity in computing, Proceedings of the 2005 Conference on. Albuquerque, New Mexico, USA. p. 48-50: ACM.
(Paper=76; T10=3; T11=2; T12=4: T13=1,2,3; T14=1; T15=2; T16=2; T18=NA; T19=1.1; T20/21=3; T22=2; T23=2; T24=n; T25=n)

[[77]] Ostrand, T.J., E.J. Weyuker, and R.M. Bell (2007). Automating algorithms for the identification of fault-prone files in Software testing and analysis, Proceedings of the 2007 International symposium on London, United Kingdom. p. 219-227: ACM.
(Paper=77; T10=3; T11=4; T12=4: T13=2,3; T14=1; T15=2; T16=2; T18=NA; T19=1, 2; T20/21=3,7; T22=2; T23=2; T24=n; T25=n)

[[78]] Ostrand, T.J., E.J. Weyuker, R.M. Bell, and R.C.W. Ostrand (2005). A different view of fault prediction in Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International. p3-4
(Paper=78; T10=3; T11=2; T12=4: T13=1,2; T14=NA; T15=2; T16=2; T18=NA; T19=1.1; T20/21=3; T22=2; T23=2; T24=n; T25=n)

[[79]] Padberg, F. (2002). Empirical interval estimates for the defect content after an inspection in Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on p. 58-68.
(Paper=79; T10=4; T11=1; T12=8: T13=5; T14=NA; T15=2; T16=5; T18=NA; T19=NA; T20/21=3,4; T22=2; T23=2; T24=n; T25=y)

[[80]] Padberg, F., T. Ragg, and R. Schoknecht (2004). Using machine learning for estimating the defect content after an inspection. Software Engineering, IEEE Transactions on. 30(1): p. 17-28.
(Paper=80; T10=2; T11=1; T12=8: T13=5; T14=NA; T15=2; T16=3; T18=1 3; T19=Not clear; T20/21=4; T22=2; T23=2; T24=n; T25=y)

[[81]] Pai, G.J. and J.B. Dugan (2007). Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods. Software Engineering, IEEE Transactions on. 33(10): p. 675-686.
(Paper=81; T10=2; T11=1; T12=2: T13=3,5; T14=2; T15=3; T16=1; T18=1; T19=NA; T20/21=Not clear; T22=1; T23=2; T24=n; T25=y)

[[82]] Pelayo, L. and S. Dick (2007). Applying Novel Resampling Strategies To Software Defect Prediction in North American Fuzzy Information Processing Society, 2007. NAFIPS '07. Annual Meeting of the. p. 69-72.
(Paper=82; T10=2; T11=1; T12=1: T13=3; T14=2; T15=1; T16=1; T18=5; T19=NA; T20/21=Not clear; T22=2; T23=1; T24=n; T25=y)

[[83]] Pighin, M. and A. Marzona (2003). An Empirical Analysis of Fault Persistence Through Software Releases in Empirical Software Engineering, Proceedings of the 2003 International Symposium: IEEE Computer Society.
(Paper=83; T10=3; T11=1; T12=4: T13=1, 2,3; T14=4; T15=2; T16=2; T18=NA; T19=5; T20/21=3,7; T22=2; T23=2; T24=n; T25=n)

[[84]] Qinbao, S., M. Shepperd, M. Cartwright, and C.A.M.C. Mair (2006). Software defect association mining and defect correction effort prediction. Software Engineering, IEEE Transactions on. 32(2): p. 69-82.
(Paper=84; T10=2; T11=1; T12=2: T13=1,2; T14=NA; T15=1; T16=1; T18=7; T19=NA; T20/21=1,2; T22=2; T23=2; T24=y; T25=y)

[[85]] Rodriguez, D., R. Ruiz, J. Cuadrado-Gallego, and J. Aguilar-Ruiz (2007). Detecting Fault Modules Applying Feature Selection to Classifiers in Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on. p. 667-672.
(Paper=85; T10=2; T11=1; T12=1: T13=1,3; T14=2; T15=1; T16=1; T18=2 7; T19=NA; T20/21=1; T22=2; T23=1; T24=y; T25=y)

[[86]] Schroter, A., T. Zimmermann, R. Premraj, and A. Zeller (2006). Where do bugs come from? ACM Sigsoft Software Eng. Notes. 31(6): p. 1-2.
(Paper=86; T10=1; T11=2; T12=2: T13=1,2,3; T14=5; T15=2; T16=2; T18=NA; T19=3; T20/21=6; T22=2; T23=2; T24=n; T25=n)

[[87]] Schroter, A., T. Zimmermann, R. Premraj, and A. Zeller (2006). If your bug database could talk... (short paper) in Empirical Software Engineering, Proceedings of the 5th International Symposium on p. 18–20.
(Paper=87; T10=1; T11=2; T12=5: T13=2,3,5; T14=2; T15=3; T16=2; T18=NA; T19=3; T20/21=6; T22=2; T23=2; T24=n; T25=n)

[[88]] Schroter, A., T. Zimmermann, and A. Zeller (2006). Predicting component failures at design time in Empirical Software Engineering, Proceedings of the 5th International Symposium on. p. pages 18–27.
(Paper=88; T10=1; T11=2; T12=2: T13=1; T14=NA; T15=1; T16=3; T18=1 4 5; T19=Not clear; T20/21=1; T22=2; T23=2; T24=n; T25=y)

[[89]] Seliya, N., T.M. Khoshgoftaar, and S. Zhong (2005). Analyzing software quality with limited fault-proneness defect data in High-Assurance Systems Engineering (HASE'05), Proc. of Ninth IEEE International Symposium on p. 89–98.
(Paper=89; T10=2; T11=1; T12=1: T13=3; T14=2; T15=1; T16=1; T18=2 7; T19=NA; T20/21=2; T22=2; T23=2; T24=n; T25=n)

[[90]] Sherriff, M., S.S. Heckman, M. Lake, and L. Williams (2007). Identifying fault-prone files using static analysis alerts through singular value decomposition in Center for advanced studies on Collaborative research, Proceedings of the 2007 Conference of the Richmond Hill, Ontario, Canada. p. 276-279: ACM.
(Paper=90; T10=3; T11=5; T12=4: T13=1,2,3; T14=2; T15=1; T16=6; T18=NA; T19=NA; T20/21=1,2; T22=2; T23=2; T24=n; T25=n)

[[91]] Sherriff, M., N. Nagappan, L. Williams, and M. Vouk (2005). Early estimation of defect density using an in-process Haskell metrics model in Advances in model-based testing, Proceedings of the 1st International workshop on St. Louis, Missouri. p. 1-6: ACM.
(Paper=91; T10=3; T11=3; T12=3: T13=5; T14=NA; T15=2; T16=2; T18=NA; T19=1, 2, 4; T20/21=5,7; T22=2; T23=2; T24=n; T25=n)

[[92]] Śliwerski, J., T. Zimmermann, and A. Zeller (2005). When do changes induce fixes? in Mining software repositories, Proceedings of the 2005 International workshop on St. Louis, Missouri. p. 1-5: ACM.
(Paper=92; T10=1; T11=2; T12=2: T13=2; T14=NA; T15=1; T16=5; T18=NA; T19=NA; T20/21=Not clear; T22=2; T23=2; T24=n; T25=n)

[[93]] Stringfellow, C. and A. Andrews (2002). Deriving a Fault Architecture to Guide Testing. Software Quality Control. 10(4): p. 299-330.
(Paper=93; T10=3; T11=5; T12=3: T13=1; T14=NA; T15=3; T16=2; T18=NA; T19=1, 6; T20/21=Not clear; T22=2; T23=2; T24=n; T25=n)

[[94]] Succi, G., W. Pedrycz, M. Stefanovic, and J. Miller (2003). Practical Assessment of the Models for Identification of Defect-prone Classes in Object-oriented Commercial Systems Using Design Metrics. Journal of Systems and Software. 65(1): p. 1-12.
(Paper=94; T10=3; T11=1; T12=2: T13=3; T14=1,5; T15=2; T16=2; T18=NA; T19=1; T20/21=3,4; T22=2; T23=2; T24=n; T25=n)

[[95]] Sunghun, K., Pan, K., and Whitehead, E. E. (2006). Memories of bug fixes in Foundations of Software Engineering, Proceedings of the 14th ACM SIGSOFT International Symposium on Portland, USA. p35-45
(Paper=95; T10=1; T11=2; T12=3: T13=1,2; T14=NA; T15=1; T16=1; T18=2 7; T19=NA; T20/21=2; T22=2; T23=2; T24=n; T25=n)

[[96]] Thwin, M.M.T. and T.-S. Quah (2002). 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. p. 2312-2316.
(Paper=96; T10=3; T11=5; T12=7: T13=3; T14=4,5; T15=4; T16=3; T18=1 3; T19=1; T20/21=4,5,6; T22=2; T23=2; T24=n; T25=y)

[[97]] Tomaszewski, P., H. Grahn, and L. Lundberg (2006). A Method for an Accurate Early Prediction of Faults in Modified Classes in Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference p487-496.
(Paper=97; T10=3; T11=1; T12=2: T13=2; T14=NA; T15=3; T16=5; T18=7; T19=NA; T20/21=Not clear; T22=2; T23=2; T24=n; T25=n)

[[98]] Turhan, B. and A. Bener (2007). A Multivariate Analysis of Static Code Attributes for Defect Prediction in Quality Software, 2007. QSIC '07. Seventh International Conference on p. 231-237.
(Paper=98; T10=2; T11=1; T12=1: T13=3; T14=2; T15=3; T16=1; T18=1 7; T19=NA; T20/21=1; T22=2; T23=2; T24=n; T25=y)

[[99]] Vivanco, R. (2007). Improving Predictive Models of Software Quality Using an Evolutionary Computational Approach in Software Maintenance, 2007. ICSM 2007. IEEE International Conference on. p. 503-504.
(Paper=99; T10=3; T11=2; T12=2: T13=3; T14=2; T15=3; T16=3; T18=1 7; T19=Not clear; T20/21=Not clear; T22=2; T23=2; T24=n; T25=y)

[[100]] Weyuker, E., J. , T. Ostrand, J. , and R. Bell, M. (2007). Using Developer Information as a Factor for Fault Prediction in Predictor Models in Software Engineering, Proceedings of the Third International Workshop on IEEE Computer Society.
(Paper=100; T10=3; T11=5; T12=4: T13=2; T14=NA; T15=3; T16=2; T18=NA; T19=1.1; T20/21=1; T22=2; T23=2; T24=n; T25=n)

[[101]] Williams, C.C. (2005). Automatic Mining of Source Code Repositories to Improve Bug Finding Techniques. Software Engineering, IEEE Transactions on. 31(6): p. 466-480.
(Paper=101; T10=1; T11=1; T12=3: T13=2; T14=NA; T15=3; T16=1; T18=1 7; T19=NA; T20/21=2,1; T22=2; T23=2; T24=n; T25=n)

[[102]] Wohlin, C., M. Host, and M.C. Ohlsson (2000). Understanding the sources of software defects: a filtering approach in Program Comprehension, 2000. Proceedings. IWPC 2000. 8th International Workshop. p. 9-17.
(Paper=102; T10=3; T11=3; T12=8: T13=3,5; T14=2; T15=2; T16=3; T18=7; T19=Not clear; T20/21=6; T22=2; T23=2; T24=n; T25=n)

[[103]] Xing, F., P. Guo, and M.R. Lyu (2005). A novel method for early software quality prediction based on support vector machine in Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on. p. 10 pp.

(Paper=103; T10=3; T11=3; T12=1: T13=3,5; T14=5,7; T15=1; T16=1; T18=4; T19=NA; T20/21=2; T22=2; T23=2; T24=n; T25=y)

[[104]] Yu, P., T. Systa, and H. Muller (2002). Predicting Fault-Proneness Using OO Metrics: An Industrial Case Study in European Conference Software Maintenance and Reeng. (CSMR 2002), Proc. Sixth p. 99-107.
(Paper=104; T10=3; T11=1; T12=1: T13=3; T14=2; T15=3; T16=2; T18=NA; T19=1, 2, 3; T20/21=5,7; T22=2; T23=2; T24=n; T25=n)

[[105]] Zhang, H. and X. Zhang (2007). Comments on "Data Mining Static Code Attributes to Learn Defect Predictors". Software Engineering, IEEE Transactions on. 33(9): p. 635-637.
(Paper=105; T10=2; T11=4; T12=1: T13=3; T14=2; T15=1; T16=1; T18=1; T19=NA; T20/21=1; T22=2; T23=2; T24=y; T25=n)

[[106]] Zhiwei, X., T.M. Khoshgoftaar, and E.B. Allen (2000). Prediction of software faults using fuzzy nonlinear regression modeling in High Assurance Systems Engineering, 2000, Fifth IEEE International Symposim on. HASE 2000. p. 281-290.
(Paper=106; T10=3; T11=3; T12=1: T13=3,5; T14=2; T15=2; T16=1; T18=3; T19=NA; T20/21=3,4; T22=2; T23=2; T24=n; T25=y)

[[107]] Zhou, Y. and H. Leung (2006). Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. Software Engineering, IEEE Transactions on. 32(10): p. 771-789.
(Paper=107; T10=2; T11=1; T12=2: T13=3; T14=1,5; T15=3; T16=3; T18=1,6; T19=1; T20/21=1; T22=1; T23=1; T24=y; T25=y)

[[108]] Zimmermann, T. and N. Nagappan (2007). Predicting Subsystem Failures using Dependency Graph Complexities in Software Reliability, 2007. ISSRE '07. The 18th IEEE International Symposium. p. 227-236.
(Paper=108; T10=3; T11=3; T12=5: T13=3; T14=4; T15=2; T16=2; T18=NA; T19=1,3,4; T20/21=7; T22=2; T23=2; T24=n; T25=y)

[[109]] Zimmermann, T., R. Premraj, and A. Zeller (2007). Predicting Defects for Eclipse in Predictor Models in Software Engineering, Proceedings of the Third International Workshop on IEEE Computer Society.
(Paper=109; T10=1; T11=2; T12=2: T13=1,3; T14=4; T15=3; T16=2; T18=NA; T19=1, 3; T20/21=5,1,6,2; T22=2; T23=2; T24=y; T25=n)

[[110]] Gondra, I. 2008. Applying machine learning to software fault-proneness prediction. *J. Syst. Softw.* 81, 2 (Feb. 2008), 186-195.
(Paper=110; T10=2; T11=1; T12=1: T13=3; T14=2; T15=1; T16=1; T18=3,4; T19=NA; T20/21=1; T22=2; T23=2; T24=y; T25=y)

[[111]] D'Ambros, M., Lanza, M., and Robbes, R. 2009. On the Relationship Between Change Coupling and Software Defects. In *Proceedings of the 2009 16th Working Conference on Reverse Engineering* (October 13 - 16, 2009). WCRE. IEEE Computer Society
(Paper=111; T10=1; T11=2; T12=2: T13=2; T14=NA; T15=2; T16=2; T18=NA; T19=1; T20/21=5,7; T22=2; T23=2; T24=n; T25=y)

[[112]] Kastro, Y. and Bener, A. B. 2008. A defect prediction method for software versioning. *Software Quality Journal* 16, 4 (Dec. 2008), 543-562
(Paper=112; T10=1; T11=5; T12=9: T13=1,2; T14=NA; T15=1; T16=1; T18=3; T19=NA; T20/21=1; T22=2; T23=2; T24=y; T25=n)

[[113]] Turhan, B., Kocak, G., and Bener, A. 2009. Data mining source code for locating software bugs: A case study in telecommunication industry. *Expert Syst. Appl.* 36, 6 (Aug. 2009), 9986-9990
(Paper=113; T10=4; T11=5; T12=1: T13=3; T14=3,4; T15=1; T16=1; T18=1; T19=NA; T20/21=8; T22=2; T23=2; T24=n; T25=n)

[[114]] Elish, K. O. and Elish, M. O. 2008. Predicting defect-prone software modules using support vector machines. *J. Syst. Softw.* 81, 5 (May. 2008), 649-660.
(Paper=114; T10=2; T11=1; T12=1: T13=3; T14=2; T15=1; T16=1; T18=4; T19=NA; T20/21=1,7; T22=2; T23=2; T24=n; T25=y)

[[115]] Pendharkar, P. C. 2010. Exhaustive and heuristic search approaches for learning a software defect prediction model. *Eng. Appl. Artif. Intell.* 23, 1 (Feb. 2010), 34-40, available on-line in 2009 – hence inclusion
(Paper=115; T10=2; T11=1; T12=1: T13=3; T14=2; T15=1; T16=1; T18=3, 5; T19=NA; T20/21=9; T22=2; T23=2; T24=n; T25=y)

[[116]] Kanmani, S., Uthariaraj, V. R., Sankaranarayanan, V., and Thambidurai, P. 2007. Object-oriented software fault prediction using neural networks. *Inf. Softw. Technol.* 49, 5 (May. 2007), 483-492.
(Paper=116; T10=5; T11=1; T12=2: T13=3; T14=1,3; T15=1; T16=3; T18=3; T19=1; T20/21=1,2; T22=1; T23=0; T24=y; T25=y)

[[117]] Turhan, B., Kocak, G., and Bener, A. 2008. Software Defect Prediction Using Call Graph Based Ranking (CGBR) Framework. In *Proceedings of the 2008 34th Euromicro Conference Software Engineering and Advanced Applications* (September 03 - 05, 2008).
(Paper=117; T10=3; T11=1; T12=1: T13=3; T14=1,3; T15=1; T16=1; T18=1; T19=NA; T20/21=1,2; T22=1; T23=2; T24=n; T25=y)

[[118]] Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M., and Haesen, R. 2008. Mining software repositories for comprehensible software fault prediction models. *J. Syst. Softw.* 81, 5 (May. 2008), 823-839.
(Paper=118; T10=2; T11=1; T12=1: T13=3; T14=2; T15=1; T16=1; T18=7; T19=NA; T20/21=1,7; T22=2; T23=1; T24=y; T25=y)

[[119]] Rana, Z. A., Shamail, S., and Awais, M. M. 2009. Ineffectiveness of Use of Software Science Metrics as Predictors of Defects in Object Oriented Software. In *Proceedings of the 2009 WRI World Congress on Software Engineering - Volume 04* (May 19 - 21, 2009). WCSE. IEEE Computer Society
(Paper=119; T10=2; T11=1; T12=1: T13=3; T14=4; T15=1; T16=1; T18=1,5; T19=NA; T20/21=1; T22=1; T23=2; T24=y; T25=n)

[[120]] Mende, T., Koschke, R., and Leszak, M. 2009. Evaluating Defect Prediction Models for a Large Evolving Software System. In *Proceedings of the 2009 European Conference on Software Maintenance and Reengineering* (March 24 - 27, 2009). CSMR. IEEE Computer Society
(Paper=120; T10=3; T11=1; T12=4: T13=2,3; T14=4; T15=1; T16=1; T18=6; T19=NA; T20/21=1; T22=2; T23=2; T24=y; T25=n)

[[121]] Calikli, G.; Tosun, A.; Bener, A.; Celik, M.; 2009. The effect of granularity level on software defect prediction. Computer and Information Sciences. ISCIS 2009. 24th International Symposium on 14-16 Sept. 2009 Page(s):531 - 536
(Paper=121; T10=1; T11=1; T12=7: T13=3; T14=1,2,3,4; T15=1; T16=1; T18=1; T19=NA; T20/21=1,9; T22=2; T23=2; T24=n; T25=y)

[[122]] Selvarani, R.; Nair, T.R.G.; Prasad, V.K. 2009. Estimation of Defect Proneness Using Design Complexity Measurements in Object-Oriented Software. International Conference on Signal Processing Systems, 15-17 May 2009 Page(s):766 – 770
(Paper=122; T10=3; T11=2; T12=7: T13=3; T14=4; T15=1; T16=6; T18=NA; T19=NA; T20/21=8; T22=2; T23=2; T24=n; T25=n)

[[123]] Wong, W. E., Horgan, J. R., Syring, M., Zage, W., and Zage, D. 2000. Applying design metrics to predict fault-proneness: a case study on a large-scale software system. *Softw. Pract. Exper.* 30, 14, 1587-1608.
(Paper=123; T10=3; T11=1; T12=8: T13=3; T14=3; T15=1; T16=2; T18=NA; T19=5; T20/21=1; T22=2; T23=2; T24=n; T25=n)

[[124]] Singh, Y., Kaur, A., and Malhotra, R. 2008. Predicting Software Fault Proneness Model Using Neural Network. In *Proceedings of the 9th international Conference on Product-Focused Software Process Improvement* (Monte Porzio Catone, Italy, June 23 - 25, 2008). A. Jedlitschka and O. Salo, Eds. Lecture Notes in Computer Science, vol. 5089. Springer-Verlag, Berlin, Heidelberg, 204-214.
(Paper=124; T10=2; T11=3; T12=7: T13=3; T14=3,4; T15=1; T16=3; T18=3; T19=1; T20/21=1; T22=2; T23=0; T24=n; T25=y)

[[125]] Guo, P., & Lyu, M. R. (2000). Software quality prediction using mixture models with EM algorithm. In First Asia-Pacific Conference on Quality Software (pp. 69–80). Hong Kong, China: IEEE Computer Society.
(Paper=125; T10=3; T11=3; T12=1: T13=3; T14=1,4; T15=1; T16=2; T18=NA; T19=6; T20/21=2,4; T22=2; T23=2; T24=n; T25=n)

[[126]] Schneidewind, N. F. (2001). Investigation of logistic regression as a discriminant of software quality. In Seventh international symposium on software metrics (328–337). Washington, DC: IEEE Computer Society.
(Paper=126; T10=3; T11=5; T12=1: T13=3; T14=1,2,4; T15=1; T16=2; T18=NA; T19=1; T20/21=2,4; T22=2; T23=2; T24=n; T25=n)

[[127]] Yuan, X., Khoshgoftaar, T. M., Allen, E. B., & Ganesan, K. (2000). An application of fuzzy clustering to software quality prediction. In Third IEEE symposium on application-specific systems and software engineering technology (p. 85). Washington, DC: IEEE Computer Society.
(Paper=127; T10=3; T11=5; T12=1: T13=3; T14=1,2,3; T15=1; T16=2; T18=NA; T19=1; T20/21=1,2; T22=2; T23=2; T24=n; T25=n)

[[128]] Khoshgoftaar, T. M., Allen, E. B., & Busboom, J. C. (2000). Modeling software quality: The software measurement analysis and reliability toolkit. In Twelfth IEEE international conference on tools with artificial intelligence (pp. 54–61). Vancouver, BC, Canada: IEEE Computer Society.
(Paper=128; T10=3; T11=3; T12=1: T13=3; T14=1,3; T15=1; T16=3; T18=5; T19=6; T20/21=2; T22=2; T23=2; T24=n; T25=y)

[[129]] Khoshgoftaar, T. M., Geleyn, E., & Gao, K. (2002). An empirical study of the impact of count models predictions on module-order models. In Eighth international symposium on software metrics (pp. 161–172). Ottawa, Canada: IEEE Computer Society.
(Paper=129; T10=3; T11=1; T12=1: T13=3; T14=1,2; T15=2; T16=2; T18=NA; T19=6; T20/21=4; T22=2; T23=2; T24=n; T25=y)

[[130]] Khoshgoftaar, T. M. (2002). Improving usefulness of software quality classification models based on boolean discriminant functions. In Thirteenth international symposium on software reliability engineering (pp. 221–230). Annapolis, MD, USA: IEEE Computer Society.
(Paper=130; T10=3; T11=3; T12=8: T13=3; T14=1,2,4; T15=1; T16=2; T18=NA; T19=6; T20/21=2; T22=2; T23=2; T24=n; T25=n)

[[131]] Khoshgoftaar, T. M., & Seliya, N. (2002). Software quality classification modeling using the SPRINT decision tree algorithm. In Fourth IEEE international conference on tools with artificial intelligence (pp. 365–374). Washington, DC: IEEE Computer
(Paper=131; T10=3; T11=5; T12=1: T13=3; T14=1,2,3,4; T15=1; T16=1; T18=5; T19=NA; T20/21=1,2; T22=2; T23=2; T24=y; T25=n)

[[132]] Khoshgoftaar, T. M., Seliya, N., & Gao, K. (2005). Assessment of a new three-group software quality classification technique: An empirical case study. Empirical Software Engineering, 10(2), 183–218.

(Paper=132; T10=3; T11=1; T12=1: T13=3; T14=1,2,3,4; T15=1; T16=3; T18=5; T19=6; T20/21=2; T22=2; T23=2; T24=n; T25=y)

[[133]] Pizzi, N. J., Summers, R., & Pedrycz, W. (2002). Software quality prediction using median-adjusted class labels. In International joint conference on neural networks (pp. 2405–2409). Honolulu, HI: IEEE Comp Soc
(Paper=133; T10=5; T11=4; T12=2: T13=3; T14=1,2,3,4; T15=1; T16=1; T18=3; T19=NA; T20/21=1,2; T22=1; T23=2; T24=y; T25=n)

[[134]] Reformat, M. (2003). A fuzzy-based meta-model for reasoning about number of software defects. In Tenth international fuzzy systems association world congress, Istanbul, Turkey (pp. 644–651).
(Paper=134; T10=3; T11=4; T12=1: T13=3; T14=1,2,3,4; T15=1; T16=1; T18=3; T19=NA; T20/21=1; T22=2; T23=2; T24=n; T25=y)

[[135]] Menzies, T., Turhan, B., Bener, A., Gay, G., Cukic, B., and Jiang, Y. 2008. Implications of ceiling effects in defect predictors. In *Proceedings of the 4th international Workshop on Predictor Models in Software Engineering* (Leipzig, Germany, May 12 - 13, 2008). PROMISE '08.
(Paper=135; T10=2; T11=1; T12=1: T13=3; T14=1,2,4; T15=1; T16=1; T18=1, 5; T19=NA; T20/21=1; T22=2; T23=1; T24=n; T25=y)

[[136]] Mahaweerawat, A., Sophasathit, P., & Lursinsap, C. (2002). Software fault prediction using fuzzy clustering and radial basis function network. International conference on intelligent technologies, Vietnam (p304–313).
(Paper=136; T10=6; T11=5; T12=9: T13=3; T14=1,2,3; T15=1; T16=1; T18=2,3; T19=NA; T20/21=1; T22=2; T23=2; T24=y; T25=n)

[[137]] Mahaweerawat, A., Sophatsathit, P., Lursinsap, C., & Musilek, P. (2004). Fault prediction in object-oriented software using neural network techniques. In Proceedings of the InTech conference, Houston, pp. 27–34
(Paper=137; T10=1; T11=1; T12=2: T13=3; T14=1,2,3,4; T15=1; T16=1; T18=3; T19=NA; T20/21=1,2; T22=1; T23=1; T24=y; T25=n)

[[138]] Mahaweerawat, A., Sophatsathit, P., & Lursinsap, C. (2007). Adaptive self-organizing map clustering for software fault prediction. In Fourth international joint conference on computer science and software engineering, Khon Kaen, Thailand (pp. 35–41).
(Paper=138; T10=6; T11=5; T12=1: T13=3; T14=1,2,4; T15=1; T16=1; T18=3; T19=NA; T20/21=1; T22=2; T23=2; T24=y; T25=n)

[[139]] Menzies, T., & Di Stefano, J. S. (2004). How good is your blind spot sampling policy? In Eighth IEEE international symposium on high-assurance systems engineering (pp. 129–138). Tampa, FL, USA: IEEE
(Paper=139; T10=2; T11=1; T12=1: T13=3; T14=1,2,4; T15=1; T16=3; T18=5; T19=1; T20/21=1; T22=2; T23=2; T24=y; T25=n)

[[140]] Zhong, S., Khoshgoftaar, T. M., Seliya, N. (2004). Unsupervised learning for expert-based software quality estimation. In Eighth IEEE international symposium on high assurance systems engineering (p149–155).
(Paper=140; T10=2; T11=1; T12=1: T13=3; T14=1,2,4; T15=1; T16=3; T18=2,3; T19=Not clear; T20/21=1,2; T22=2; T23=2; T24=n; T25=y)

[[141]] Mertik, M., Lenic, M., Stiglic, G., & Kokol, P. (2006). Estimating software quality with advanced data mining techniques. In International conference on software engineering advances (p.19). Papeete, Tahiti, French Polynesia: IEEE Computer Society.
(Paper=141; T10=2; T11=5; T12=1: T13=3; T14=1,2,4; T15=1; T16=1; T18=4,5,7; T19=NA; T20/21=1; T22=2; T23=2; T24=y; T25=n)

[[142]] Boetticher, G. (2006). Improving credibility of machine learner models in software engineering. Advanced machine learner applications in software engineering. Series on software engineering and knowledge engineering. Hershey, PA, USA: Idea Group Publishing.
(Paper=142; T10=2; T11=1; T12=1: T13=3; T14=1,2,4; T15=1; T16=3; T18=1,2,5; T19=Not clear; T20/21=1,2; T22=1; T23=2; T24=y; T25=y)

[[143]] Yang, B., Yao, L., & Huang, H. Z. 2007. Early software quality prediction based on a fuzzy neural network model. In Third international conference on natural computation, Haikou, Çin (pp. 760–764).
(Paper=143; T10=6; T11=5; T12=1: T13=3; T14=1,2,3,4; T15=1; T16=1; T18=3; T19=NA; T20/21=8; T22=2; T23=2; T24=n; T25=n)

[[144]] Turhan, B., Menzies, T., Bener, A. B., and Di Stefano, J. 2009. On the relative value of cross-company and within-company data for defect prediction. *Empirical Softw. Engg.* 14, 5
(Paper=144; T10=4; T11=4; T12=1: T13=3; T14=1,2,4; T15=1; T16=3; T18=1,2; T19=Not clear; T20/21=1; T22=2; T23=2; T24=n; T25=y)

[[145]] Weyuker, E. J., Ostrand, T. J., and Bell, R. M. 2008. Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models. *Empirical Softw. Engg.* 13, 5 (Oct. 2008), 539-559.
(Paper=145; T10=3; T11=4; T12=1: T13=2,3; T14=1,2,4; T15=1; T16=2; T18=NA; T19=1; T20/21=8; T22=2; T23=2; T24=n; T25=n)

[[146]] Jiang, Y., Cukic, B., and Ma, Y. 2008. Techniques for evaluating fault prediction models. *Empirical Softw. Engg.* 13, 5 (Oct. 2008), 561-595.
(Paper=146; T10=2; T11=4; T12=1: T13=3; T14=1,2,4; T15=1; T16=3; T18=1,2,6; T19=1; T20/21=1,4,7; T22=1; T23=2; T24=y; T25=y)

[[147]] Khoshgoftaar, T. M., Yuan, X., Allen, E. B., Jones, W. D., and Hudepohl, J. P. 2002. Uncertain Classification of Fault-Prone Software Modules. *Empirical Softw. Engg.* 7, 4 (Dec. 2002), 297-318
(Paper=147; T10=3; T11=3; T12=1: T13=3; T14=1,2,4; T15=1; T16=1; T18=5; T19=NA; T20/21=2; T22=2; T23=2; T24=y; T25=n)

[[148]] Khoshgoftaar, T. M., Seliya, N., and Gao, K. 2005. Assessment of a New Three-Group Software Quality Classification Technique: An Empirical Case Study. *Empirical Softw. Engg.* 10, 2 (Apr. 2005), 183-218.
(Paper=148; T10=3; T11=3; T12=1: T13=3; T14=1,2,4; T15=1; T16=1; T18=5; T19=NA; T20/21=2; T22=2; T23=2; T24=y; T25=n)

Appendices

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: Publication Sources

Journals (where more than 1 study has been published)	**Number of included studies**
IEEE Transactions on Software Engineering	17
Empirical Software Engineering	5
Journal of Systems and Software	5
Journals where only 1 study published	11
Total	38
Proceedings (where more than 1 study has been published)	
Automated Software Engineering, Int'l conference on	4
Computer Software and Applications Conference (COMPSAC), Int'l conference	2
Empirical Software Engineering and measurement, Int' Symposium on	8
European Conference Software Maintenance and Reeng. (CSMR), Proc. of	3
Euromicro conference	3
Foundations of SE, Int'l symposium on	2
High-Assurance Systems Engineering (HASE), International Symposium on	4
Mining software repositories, International workshop on	3
Neural networks (IJCNN), International conference on	2
Predictor Models in Software Engineering (PROMISE), Int'l workshop on	4
SIGPLAN SIGSOFT	10
Software Engineering, Int'l Conference (ICSE)	10
Software Maintenance, Int'l Conf on (ICSM)	3
Software Reliability Engineering, Int'l conference on (ISSRE)	12
Software Testing and analysis, Int'l Symposium on (ISSTA)	2
Conferences where only 1 study published	38
Total	110

Appendix D. Classification of static code metrics

1	Size	LOC/size (incl comments per LOC etc)
2	General	Various general SCMs listed (often many used), typically metrics data available from NASA datasets
3	Structure	Including coupling, cohesion and inheritance, system architecture, CK metrics
4	Complexity	Includes Halstead and McCabe
5	Other	Includes code clones (code fragments and language constructs) variables/types of statements (e.g. global variables, incoming and outgoing variable accesses, 'var' statements, 'retrieve' statements

Appendix E: Statistics used in fault prediction studies

Statistical approach	Approach includes
1. Regression	• Negative binomial. • Poisson regression/zero-inflated Poisson regression, Loglinear regression, Bernoulli regression, regularised logistic regression, logistic regression (including stepwise and exact), binary logistic regression, risk coefficient • Linear regression, generalized linear regression, multiple regression, multiple linear regression, • Non-linear regression, non linear least squares regression • logarithmic, exponential models • Univariate modelling • Multivariate modelling
2. Compare means/variances/ significance test	t-statistic, t-test, ANOVA, F-Test, discriminant analysis, variance inflation factor, chi-square
3. Correlation	Spearman/Spearman's rank correlation, Pearson correlation/Pearson product-moment correlation, Kendall's rank correlation, Bravais, scatterplots,
4. Feature/model selection	Principal component analysis (PCA), Akaike Information Criterion
5. Descriptive stats	Mean, standard deviation, confidence interval, median, inter-quartile range, percentage
6. Other	Impact analysis, pairwise intersections, Cox proportional hazards, Varimax rotation, Alberg diagram

Appendix F: Lessman classification scheme used to classify type of machine learning approach used in studies

Classification model	Approaches included
1. Statistical classifiers	• Linear Discriminant Analysis • Quadratic Discriminant Analysis • Logistic Regression • Naïve Bayes • Bayesian Networks • Least-Angle Regression • Relevance Vector Machine
2. Nearest neighbour methods	• *k*Nearest Neighbour • K-Nearest Neighbour
3. Neural networks	• Multi-Layer Perceptron • Radial Basis Function Network
4. Support vector machine-based	• Support vector machine • Lagrangian • Least Squares SVN • Linear Programming • Voted Perceptron
5. Decision tree approaches	• C 4.5 Decision Tree • Classification and Regression Tree • Alternating Decision Tree
6. Ensemble methods	• Random Forest • Logistic Model Tree

Appendix G: Classification of how studies balance data

Data balancing used		
Approach		**Description**
0	Data already balanced	The data set contained a balanced set of faulty as opposed to non-faulty modules
1	True balancing of data	Use of over/under sampling methods, SMOTE, etc to achieve a 50/50 split of faulty v non-faulty code units
2	No explicit data balancing done	Imbalanced data used; no balancing reported; statistical techniques applied

Appendix H. Classification of how categorical studies have measured model performance

Performance Indicator	Examples
1. Confusion Matrix related composite measures	F-measure, Recall, Precision, Accuracy, misclassification rate/error-rate, Sensitivity, Specificity, pd, pf, Balance, correctness, completeness, AUC (ROC)
2. Confusion Matrix constructs	Rates of: FP, TN, FN, TP Type I = false positive (FP); Type II = false negative (FN)

Appendix I. Classification of how continuous studies have measured model performance

Performance Indicator	Examples
Error rates	MRE, MAE, PRED, standard error, absolute error, jack-knife error etc)
Regression coefficient	Best R ~/R^2
Correlation test	Pearson, Spearman
Variance significance test	t-test, F-test, goodness of fit, chi square, p-value
Other	Theil forecasting stat