Fault Detection and Prediction in an Open-Source Software Project

Michael English
Lero
CSIS Department
University of Limerick, Ireland
Michael.English@lero.ie

Chris Exton
CSIS Department
University of Limerick, Ireland
Chris.Exton@ul.ie

Irene Rigon
Lero
CSIS Department
University of Limerick, Ireland
irenerigon@gmail.com

Brendan Cleary
CSIS Department
University of Limerick, Ireland
Brendan.Cleary@ul.ie

ABSTRACT

Software maintenance continues to be a time and resource intensive activity. Any efforts that help to address the maintenance bottleneck within the software lifecycle are welcome. One area where such efforts are useful is in the identification of the parts of the source-code of a software system that are most likely to contain faults and thus require changes. We have carried out an empirical study where we have merged information from the CVS repository and the Bugzilla database for an open-source software project to investigate whether or not parts of the source-code are faulty, the number and severity of faults and the number and types of changes associated with parts of the system. We present an analysis of this information, showing that Pareto's Law holds and we evaluate the usefulness of the Chidamber and Kemerer metrics for identifying the fault-prone classes in the system analysed.

General Terms

Measurement, Software quality

Keywords

empirical study, metrics, fault prediction, open source, regression

1. INTRODUCTION

The development of large software systems is a time and resource intensive activity. Even with the increasing automation of software development activities, resources are still scarce. Therefore, we need to be able to provide accurate information and guidelines to managers to help them make decisions, plan and schedule activities, and allocate resources for the different software activities that take place during software development. Software measurement is necessary to assist this decision making process before and during the software development lifecycle.

Testing of large systems is an example of a resource and time-consuming activity. Applying equal testing and verification effort to all parts of a software system has become cost-prohibitive [2]. Therefore, identifying parts of the software where testing efforts should be focused can help software engineers and project managers, both now and in the future, to direct peer-reviews, testing, inspections, and restructuring efforts towards these critical parts of the software. As a result, developers can use their resources more efficiently to deliver higher quality products in a timely manner.

Due to the nature of Open Source software the web has in effect become a shared distributed development forum for all aspects of software design and maintenance. In an effort to leverage this fact we have implemented a research method broadly based on a Content Analysis [41] based approach. Most researchers would accept that quantification is an essential component of research. It is necessary if we hope to test the relative relationships between selected variables. One of the main strengths of Content Analysis compared to other methods is that it can make use of information sources that are not normally exploited by existing methods and provides a means to use quantitative analysis on what would normally be considered qualitative data. Analysis on such a data source not only produces quantitative data but also at the same time provides a good degree of rigor, replication and experimental control.

Using such sources such as open sources repositories also opens up the possibility of gaining 'real life' data and as such promises a greater level of ecological validity and insights into how Software Engineers interact in their other environments when compared to studies based solely on materials and situations manufactured by researchers.

In addition as the open source community has records spanning back over a number of years so it becomes possible to implement longitudinal studies, which are notoriously difficult in the fluctuating and fast moving world of Software

Engineering.

The CVS and Bugzilla databases for an open source project are used as the data sources for the empirical study reported in this paper. In particular, in this paper we consider the evolution of the source code of a software system and the usefulness of the Chidamber and Kemerer software metrics [16] for predicting fault-prone parts of the system studied.

2. RELATED WORK

Pareto's Law (also known as the 80:20 rule) states that 80% of the effects can be contributed to 20% of the causes. It is a principle that seems to holds in many different contexts and was orignally used to explain income and land ownership distributions. It has also been shown to hold in the field of software engineering where a number of empirical studies have shown that a large proportion (80%) of faults and changes are centered in a small proportion (20%) of classes [40, 55]. Koru and Liu [40] showed that 80% of the changes to KOffice and Mozilla were centered in 20% of the classes. Ware *et al.* [55] examined the evolution of a commercial application over one year and found that Pareto analysis is useful for predicting change-prone classes. Given the necessity outlined in the introduction to focus testing effort on the parts of the software that are most likely to be faulty, Pareto's Law can help to focus the testing effort on these parts of the software. In this study we investigate if Pareto's Law holds for the software system analysed here.

In order to find the best ways of developing software, we must be able to control and thus improve the quality of software. To do this we must be able to measure it. As De Marco stated: "You cannot control what you cannot measure" [20]. Much of the research in software measurement has focused on building models which aim to predict some external characteristic(s) of a product by measuring internal attributes of the product and building a predictive model based on these measurements [11, 24, 32, 35, 5, 7]. In this section we will consider first the range of software metrics that have been defined and then some of the empirical studies that have utilised these metrics in assessing external quality attributes.

2.1 Analysis of Software Metrics

With the widespread acceptance of the Object-Oriented (OO) paradigm came the realisation that, while existing software metrics would be useful, they would not be sufficiently accurate to assess the use of abstractions such as encapsulation and inheritance in this new paradigm [2]. As a result many new metrics for analysing object-oriented systems have been defined. In this section, some metrics which have been proposed in the literature are presented.

One of the most common internal attributes of software products to be measured is size. Many different size measures have been proposed in the literature (Lines of code, cyclomatic complexity). These include measures of length and measures of functionality. Estimates of a product's size made at an early stage of the development process can help predict resource and cost requirements for a project.

Abreu *et al.* [1] presented a set of system-level metrics (MOOD) that return values between 0 and 1. Thus a value of 0 represents the absence of a factor and a value of 1 represents the maximum possible presence of the factor. The metrics defined by Lorenz and Kidd [42] (e.g. number of public instance methods, number of overridden methods) are more "directly countable" and thus should be easier to collect [33].

The metrics defined by Chidamber and Kemerer [16] cover many aspects of the OO paradigm and are the most commonly cited OO metrics in the literature. These metrics include:

- Weighted Methods per Class (WMC): This is the sum of the weights of all the methods in a class. If the weight assigned to each method is just one then WMC is the number of methods in the class (NMC) [18]. NMC is the metric adopted in this study.
- Depth of Inheritance Tree (DIT): The DIT of a class is the maximum distance in the inheritance tree of the class from the root node of the hierarchy.
- Number of Children (NOC): The number of children (derived classes) of a class is the number of classes inheriting directly from the class. If a class has a large number of children this suggests that this class provides a broad description of a set of classes.
- Coupling between object classes (CBO): "CBO for a class is a count of the number of other classes to which it is coupled" [16]. Two objects are coupled if "one of them acts on the other, i.e. methods of one use methods or instance variables of another ". This measure of coupling includes coupling due to inheritance. Chidamber and Kemerer state that "excessive coupling between object classes is detrimental to modular design".
- Response for a class (RFC): RFC is a measure of the magnitude of the response set for a class. The response set is composed of all the methods in the class itself and all the methods which can be called from the methods in the class.

2.2 Evaluating External Attributes of Software Systems

Many of the studies which attempt to associate internal measures with external quality attributes of systems focus on the fault-proneness of systems as an external characteristic. This stems from the fact that reliability or correctness are important characteristics in software quality models. For example, in McCall's model [44] both reliability and correctness are both categorised as quality factors. In ISO 9126 [36] one of the six sub-characteristics of internal and external quality is reliability and one of the factors for determining reliability in this model is fault-tolerance. Finally in Dromey's model [21] one of the four quality carrying properties of components of software products are correctness properties.

One way of measuring or partly measuring the reliability or correctness of components or modules (e.g. classes) in a product is to measure the number of faults or the existence of faults in these modules. If a relationship between

the internal characteristics of some modules and their fault-proneness can be established for a system, then the structure of modules with respect to these characteristics should be evaluated. If the internal attributes can be extracted from a system at an early design stage then early corrective action can be taken to improve the overall quality of a system. However, many empirical studies reported in the literature extract these metrics from the source-code of software systems.

2.2.1 Empirical studies of fault-proneness

Table 1 summarises a selection of empirical studies from the literature which attempt to build prediction models for identifying fault-prone classes or classes that are likely to have many faults in object-oriented systems. The first column of Table 1 provides the reference for where the study was reported. The second column describes the software systems that were used to provide the data for analysis. In this column 'P' stands for proprietary(a proprietary software system), 'OS' means an open-source system and 'S' means a system developed by students. The third column describes the metrics included in the analysis for the corresponding system. In this column, MOOD refers to the metrics defined by Abreu *et al.* [3] C&K means the metrics defined by Chidamber and Kemerer [16] Briand97 refers to the metrics defined by Briand *et al.* [10] and L&K stands for the Lorenz and Kidd metrics [42]. The analysis method (column four of Table 1) refers to the analysis techniques applied to establish which metrics are statistically significantly associated with faulty classes.

It is worth noting that much of the early work in validating Object-Oriented metrics utilised software systems developed by students. However, since about 2001 either proprietary or open source systems have been used to validate these metrics. These systems obviously reflect the intricacies and complexities of actual software systems. A number of researchers and research groups have utilised proprietary software systems and a range of metrics and analysis methods for their work. Others have used the data from the NASA Metrics Data Program (MDP) as the basis for their research. Surprisingly few of the papers reviewed here have utilised the potential of open-source software to carry out similar research. Two notable exception are Gyimothy *et al.* [32] who studied the Mozilla software system and Zimmermann *et al.* [58] who studied Eclipse. In this research we study a component of the Eclipse software, the Java Development Kit (JDT).

Many of the empirical studies summarised in Table 1 study the effect of the Chidamber and Kemerer metrics on fault-proneness. Coupling metrics, (CBO or RFC) and the inheritance metric, (DIT) seem to be included in many of the multivariate analysis models constructed. Similar analysis has been undertaken on the open-source software system Mozilla, [32]. The Chidamber and Kemerer metrics suite was extracted from the software. The CBO metric along with lines of code (LOC) were the best metrics for predicting the fault-proneness of classes. Subramanyam and Krishnan, [52], found similar results. In this research we also validate a subset of the Chidamber and Kemerer metrics and compare our results to those of [32].

3. THE EMPIRICAL STUDY

Perry *et al.* [49] have proposed a structure that good empirical studies should follow. Their proposed structure is embodied in this empirical study.

3.1 Research Context

In this paper we analyse the CVS and the Bugzilla database for an open-source system to link changes in the source code with faults/issues raised in Bugzilla. Previous research in this field has developed tools which exploit the information stored in version control systems such as CVS and issue tracking systems such as Bugzilla. For example, Fischer *et al.* [30], Cubranic and Murphy [19] and Sliwerski *et al.* [51] have integrated these information sources to help gain a deeper insight into the evolution of software and in particular for impact analysis purposes. As mentioned in the previous section, Gyimothy *et al.* [32] also link Bugzilla information with changes to the source code to evaluate the Chidamber and Kemerer metrics [16]. Ayari *et al.* [4] highlight some of the threats to building models from this information. Their main concerns are that many links between the two repositories cannot be recovered and also that the issues raised in an issue tracking system may not all relate to corrective maintenance and thus the resulting models may be inappropriately be called fault-detection models. Ostrand *et al.* [48] also discuss this issue. They proposed a heuristic that if more than two files were affected by a modification request then it was unlikely to be a fault because it was likely to be a change in the interface due to a change in the specification. The tool adopted in this analysis for the integration of CVS and Bugzilla information, Venus [17].

3.2 Research Questions

The following research questions are considered in this research:

- **RQ1:** Does Pareto's Law hold for the distribution of faults/issues in classes in the source code?
- **RQ2:** Are the Chidamber and Kemerer metrics [16] good predictors of faulty/non-faulty classes?
- **RQ3:** Are the Chidamber and Kemerer metrics [16] good predictors of *high*-fault prone classes?
- **RQ4:** Are the Chidamber and Kemerer metrics [16] good predictors of fault prone classes?

3.3 Study Design

In choosing a system to analyse a number of criteria needed to be taken into account. These include:

- Our study should be repeatable by others. The importance of replication of empirical studies has been extensively highlighted [13, 9, 14, 37]. Therefore, the source code of the system analysed and the experimental data should be publicly available
- The system should be of non-trivial size and thus projects developed by students should be excluded
- Access to the version control system and issue tracking system is required

Study	Data Set	Metrics	Analysis Method	Statistically Significant Metrics
[3]	S	MOOD	Pearson, MLinR	MOOD
[6]	S	C&K	ULR, MLR	CBO, RFC, DIT
[10]	S	C&K, Briand97	,ULR, MLR	OCMEC, FMMEC, DIT,RFC
[8]	S	IC,EC, inheritance and cohesion measures	LR	IC, cohesion, Inh, Ovr, Add
[12]	P (90 C++ classes)	As above	Spearman, MLR	IC(method invocation)
[11]	S		Spearman, MLR	IC(method invocation), cohesion
[22]	P(85 C++ classes)	C&K, Briand97	MLR	Size, CBO, ACMIC
[23]	P(174 C++ classes)	C&K, ⊆L&K	LR	none after controlling for size
[24]	P(69 Java classes)	C&K, Briand97		OCMEC, DIT, Size
[28]	P(2 systems)	Size, Complexity	Alberg diagrams, Negative binomial	Size(LOC)
[27]	P(31 projects)	Qualitative and Quantitative Factors	Bayesian Network	Qualitative and Quantitative
[29]	S	200 metrics	MLR	coupling cohesion and inheritance measures included
[32]	OS (Mozilla (3677 classes))	C&K	MLR, MLinR	CBO, DIT, WMC, LOC
[31]	P (145 Java classes)	⊆C&K, ⊆Briand97	MLR	IC, DIT
[34]	S(avg. 12 classes), System(12 classes), LEDA (197 classes)	C&K	Spearman	WMC
[38]	NASA MDP	Design, Code metrics	Machine Learning Algorithms	All metrics
[45]	NASA MDP	McCabe, Halstead, LOC	Machine Learning	All
[46]	P (5 Microsoft systems)	module, function and class metrics	Spearman, PCA MLinR	metrics from each group
[48]	P(2 systems)	Size,Prior faults, Age, file status, program type	Negative binomial	All
[52]	P (405 C++ classes, 301 Java classes)	⊆C&K	Pearson and WLS	Size, WMC, CBO, DIT
[53]	P(3 Systems)	C&K		WMC, RFC
[56]	123 Java classes	⊆C&K	LR	NMC, NOC, CBO_{out}, RFC_{in}, DIT, LCOM
[58]	OS (Eclipse)	method, class, file and package metrics	Spearman, LR LinR	metrics from each group

Table 1: Summary of Empirical Studies for predicting fault-proneness

Classes	1412
Files	1147
Subroutines	18198
Lines	411153
Lines Blank	28142
Lines Code	268945
Lines Comment	126895
Declarative Statements	57681
Executable Statements	127154
Ratio Comment/Code	0.47
No. of Commit operations on 05/02/2008	9874

Table 2: Summary Statistics for JDT Core component

Faults/Issues	Resolution	Total
Verified	*	4,355
Resolved	*	5,213
Other	*	866
TOTAL	*	10,433

Table 3: Issue Information for JDT Core

On this basis the Java Development Toolkit (JDT) component of the Eclipse project was chosen for this analysis. The Core sub component of the JDT comprising 1412 class files and over 268000 LOC satisfies our requirements for this experiment. Table 2 provides summary statistics for the JDT Core component.

The Bugzilla database for the Eclipse project [54] provides detailed information related to issues reported. The data in Table 3 provides details of all issues related to the Core component of the JDT product in Eclipse on the 05 February 2008.

We are interested in those issues/faults that are fixed and so we concentrate on a subset of the 10,433 (see Table 3) different issue entries. We focused on issues having the resolution marked as FIXED and status VERIFIED and RESOLVED because these are the issues that have been resolved and thus it is likely that they will be mentioned in a commit operation comment. Table 4 summarises this information. From the total of 4704 issues enumerated in Table 4, we found an impact on CVS messages of 3336 issues/faults (a 71% match). Therefore, we are able to trace the changes induced (or at least linked, to via the commit comment in CVS) by these issues/faults through the CVS system and ultimately map the issue ID with a changed piece of code and specifically a class name. We are able to map both changes and issues to code segments in this way.

Faults/Issues	Resolution	Total
Verified	Fixed	3912
Resolved	Fixed	792
TOTAL	Fixed	4704

Table 4: Issues with resolution fixed for JDT Core

Mozilla	NMC	NOC	DIT	CBO	RFC	LOC
NMC	1.00	0.00	0.16	0.43**	0.54**	0.56**
NOC		1.00	0	0	0	0
DIT			1.00	0.17	0.52**	0.08
CBO				1.00	0.48**	0.58**
RFC					1.00	0.40**
LOC						1.00
JDT	NMC	NOC	DIT	CBO	RFC	LOC
NMC	1.00	.126**	-.018	.555**	.869**	.791**
NOC		1.00	-.108	.297	.061	.047
DIT			1.00	.071	.046	-.026
CBO				1.000	.643**	.565**
RFC					1.00	.858**
LOC						1.00

Table 5: Correlations between Metrics

3.3.1 Tools

Our aim is to associate issue/fault requests that are reported in Bugzilla with specific parts of the source code of a software system. A typical Bugzilla database does not hold this type of information. However, CVS comments provide a rich source of documentation (including the issue request ID in some cases) [15].

In this work we utilise Venus [17], an Eclipse plug-in which creates traceability links between CVS and Bugzilla. Venus utilises the powerful functionality of the Eclipse platform to identify the differences between two versions of a file and thus the impact set of a change. From that impact set the resources modified (e.g. class, method) can be identified. If the corresponding CVS commit comment referred to an issue ID, then this ID can be verified using the Bugzilla database.

In addition to identifying traceability links between issues raised in Bugzilla and the source code (via CVS), it is also necessary to extract metrics from the source code of the project to help build a prediction model. The manual application of metrics to software is likely to be a time-consuming and error-prone task even for small systems. The source code analysis tool Understand for Java was used for this purpose [50].

Table 5 presents the correlations between the metrics for the JDT system (analysed in this study). and for Mozilla (analysed by Gyimothy *et al.* [32]). A ** denotes a statistically significant correlation. It is noteworthy that almost all the correlations that were statistically significant for the Mozilla study [32] are also significant for this study of the JDT system. The only exception is the significant correlation found between RFC and DIT for Mozilla which is not significant for the JDT system. Mozilla is developed in C++ and JDT in Java. We have found that C++ systems tend to have more inheritance than Java systems [25]. This may be the reason for the significant correlation in the Mozilla system here. There is also a significant correlation between NOC and NMC for JDT which doesn't exist for Mozilla. Again this highlights a possible difference between the inheritance structures in C++ systems compared to Java systems.

3.4 Data Analysis and Interpretation

20% of 1412	282
Number of faults in first 282 classes	7204
Percentage of faults in first 282 classes	82

Table 6: Pareto's Law(faults/issues in classes)

20% of 1412	282
Number of changes in first 282 classes	35244
Percentage of changes in first 282 classes	89.5

Table 7: Pareto's Law(changes in classes)

In this section we investigate each of the research questions outlined earlier.

RQ1: Does Pareto's Law hold for the distribution of faults/ issues in classes in the source code?

Table 6 provides the results for Pareto's Law. As the table illustrates 82% of faults/issues occured in 20% of the classes. This supports earlier findings outlined above, concerning Pareto's Law. Identifying the small proportion of problematic classes, and then focusing testing efforts on them could significantly improve the quality of these products, reduce the product-release cycles, and ultimately the development and maintenance costs. The priority and severity of issues has not been accounted for here but will be the focus of future work. As well as considering the number of faults/issues associated with classes, we have also considered the number of changes associated with classes as per Koru and Liu [40]. Our findings support previous results showing that in excess of 80% of changes are centered in 20% of classes.

The next stage of our analysis is to try to build a prediction model for identifying fault-prone and change-prone classes in a software system.

RQ2: Are the Chidamber and Kemerer metrics [16] good predictors of faulty/non-faulty classes?

A univariate logistic regression analysis was undertaken using some of the Chidamber and Kemerer metrics [16] along with the Lines of code (LOC) metric, each as the independent variable in the regression in turn. The dependent variable was a boolean representing whether or not classes are faulty.

JDT	NMC	NOC	DIT	CBO	LOC	RFC
Coeff.	.113	.095	.410	.165	.009	.054
Const.	-.260	.556	-.171	-.616	-.132	-.397
p-value	<.001	.005	<.001	<.001	<.001	<.001
R^2	.195	.010	.106	.323	.222	.276
Mozilla						
p-value	<.001	0.551	<.001	<.001	<.001	<.001
R^2	0.114	0.000	0.067	0.152	0.128	0.108

Table 8: Result of Univariate Logistic Regression

Table 8 presents the results of this analysis for each metric. The p-value for each metric tells whether or not this this metric is a significant predictor of the dependent variable. For each metric the p-value is significant ($< .05$), although NOC seems less significant than the other metrics. Table 8 also presents the corresponding results achieved by Gyimothy *et al.* [32]. In general our results seem to achieve considerably higher R^2 values. The NOC metric does not seem to be strongly statistically significant in either study. The coefficient reflects the strength of the impact of the impact of the independent variable. The sign of this coefficient reflects whether the impact is positive or negative. The R^2 coefficient gives the proportion of the total variation in the dependent variable that is explained by the model. The larger the value of R^2 the better the dependent variable is explained by the independent variable.

It should be noted that "in logistic regression, high R^2 values are rare opposed to the R^2 values of least-square regression because they are built on very different formulae" [32]. In this context our results are quite important since NMC, LOC, RFC and in particular CBO (R^2=32.3%) can explain a considerable amount of the variation in the dependent variable. This compares with 15.2% of the variation being explained by CBO in the study undertaken by Gyimothy *et al.* (see last two rows of Table 8).

It is likely that the metrics considered are not totally independent. If strong correlations exist between some metrics then not all of these metrics should be included in a multivariate model. The correlations between RFC and LOC and between RFC and NMC seem particularly large (see Table 5) and thus could lead to the problem of multicollinearity if all these metrics are included in a multivariate model. In fact we constructed such a model and found that NMC and LOC were not significant predictors in the model and that NMC had a negative coefficient. This is unusual since one might expect bigger classes to have more faults/issues associated with them.

We use stepwise selection to determine which variables should be included in the multivariate model. Each variable is chosen in turn as the dependent variable with all other variables as the independent variables. The R^2 value of the regression is highest when RFC is the dependent variable. Combined with the correlations above, this suggests that this metric should be excluded from the regression model. We also exclude NOC since its significance in the univariate model is less than the other metrics.

In the multivariate logistic regression analysis we found $R^2 = 0.371$. We have seen that in logistic regression $R^2 > 0.3$ is considered good so our model seems to be a good predictor. Gyimothy *et al.* [32] found that $R^2 = 0.175$ in their analysis. This result was only slightly better than the univariate model using CBO. Our findings are similar with an increase of 0.05 in R^2 in going from the univariate model (with CBO) to the multivariate model.

Logistic regression also provides classification models. These models assign each class either a faulty or non-faulty status, based on a threshold value of 0.5. This means that if the model value for a class is > 0.5 the class is classified as

JDT	Const	NMC	DIT	CBO	LOC
Coeff.	-1.155	.026	.271	.107	.002
p-value	<.001	.044	<.001	.001	.031

Table 9: Multivariate Logistic Regression

	Predicted	Predicted	Percentage
Observed	Not faulty	Faulty	Correct
Not faulty	330	157	67.8
Faulty	162(1658)	757(7132)	77.3

Table 10: Classification of Classes based on Multivariate Logistic Regression

faulty otherwise it is classified as not faulty. Each of the models (the univariate models and the multivariate model) were utilised to create a classification table.

Table 10 shows the results of the classification for the multivariate logistic regression. The numbers in parentheses are the sum of the faults that were found in that group of classes. As we can see from Table 10, the model classified 1087 (330+757) of the 1406 classes correctly, that is with the precision of 77,31%.

Table 11 shows the values for the precision, correctness, and completeness of the univariate logistic regression models and the multivariate logistic regression model.

RFC and CBO have very high correctness values (81.6% and 82.24%), meaning that only a small percentage (18.4 percent and 17.73 percent) of the faultless classes were predicted as faulty. As such they are almost as good as the multivariate model. There is little difference between the univariate models using NMC, CBO, RFC and LOC and the multivariate model from the perspective of precision. However, with regard to completeness the multivariate model is considerably better than any of the univariate models with the model using CBO as the independent variable as the next best as regards completeness. Again, in general our results suggest that these metrics are slightly better predictors than the findings of Gyimothy *et al.* [32] suggest.

RQ3: Are the Chidamber and Kemerer metrics [16] good predictors of high-fault prone classes?

Given that a large proportion of the issues/faults occur in a

Metric	Precision	Correctness	Completeness
NMC	76.30%	77.58%	64.87%
NOC	65.40%	-	-
DIT	64.79%	66.61%	53.03%
CBO	78.00%	82.24%	72.78%
RFC	76.70%	81.60%	62.35%
LOC	75.24%	76.21%	68.34%
Multi	77.31%	82.82%	81.13%

Table 11: Precision, Correctness and Completeness for Regression Models

	NMC	NOC	DIT	CBO	LOC	RFC
Coeff	.052	.083	.127	.074	.007	.035
Const	-2.2	-1.1	-1.7	-2.6	-2.7	-2.8
p-value	$< .001$	.001	.001	$< .001$	$< .001$	$< .001$
R^2	.198	.012	.012	.342	.436	.405

Table 12: Univariate Logistic Regression (classes > 10 faults/issues)

	NMC	CBO	LOC	RFC
Coefficient	.046	.037	.003	.029
p-value	$< .001$	$< .001$	$< .001$	$< .001$

Table 13: Multivariate Logistic Regression (classes > 10 faults/issues)

small proportion of classes, it is important to identify these classes that are associated with many faults. Therefore we decided to create a new dependent variable. This variable is also a boolean variable and distinguishes classes that have more than ten faults/issues associated with them from those that have less than ten faults/issues associated with them.

The univariate logistic regression with the new dependent variable is presented in Table 12. The R^2 of NOC and DIT is again very low so these metrics seem to be less useful. We exclude these two metrics from our multivariate logistic regression prediction model. In the multivariate logistic regression (see Table 13) we obtain $R^2 = 50.2\%$ illustrating that the four metrics included in the model can predict half of the variation in the dependent variable. More efforts will be expended in improving this model in the future since it is important to have a good prediction model for classes that are associated with a large number of issues/faults.

Given that logistic regression uses a binary variable as its dependent variable, we have only been able to build models for classes classified as either faulty or non-faulty. Linear regression is used for RQ4 to try to build models which utilise the number of faults as the dependent variable, since the number of faults associated with classes varies widely.

RQ4: Are the Chidamber and Kemerer metrics [16] good predictors of fault prone classes?

As in the logistic regression, the independent variables are the metrics in the linear regression, but the dependent variable is the number of faults/issues in a class.

The univariate linear regression presented in Table 14 suggests that LOC and RFC are both good predictors ($R^2 > 0.60$) of the number of faults/issues associated with a class. NMC and CBO are reasonable predictors, each explaining almost 40% of the variation in the number of issues in a class. However, both NOC and DIT are not significant predictors of the dependent variable. In fact they only explain 0.4% and 0.2% of the variation in the number of issues in a class. Our results show a much higher R^2 for LOC and RFC than the results of Gyimothy *et al.* [32] which are presented at the bottom of Table 14. Our results are slightly

JDT	NMC	NOC	DIT	CBO	LOC	RFC
Coefficient	0.396	0.463	0.332	0.476	0.024	0.210
Constant	2.178	6.921	6.584	0.668	2.213	0.097
p-value	.000	.023	.227	.000	.000	.000
R^2	.393	.004	.002	.397	.626	.607
Mozilla						
p-value	.000	0.728	.000	.000	.000	.000
R^2	0.321	0.000	0.139	0.349	0.342	0.280

Table 14: Univariate Linear Regression (Fault-proneness)

Dep. Var	R^2	F-value	p-value	Indep. Vars
Num. faults	.690	1040.7	.000	LOC,RFC,CBO

Table 15: Multivariate linear regression (Fault-proneness)

higher for NMC and CBO. The results for the inheritance metrics (DIT and NOC) are similar and while we didn;t find either metric significant, Gyimothy *et al.* found DIT to be significant although with $R^2 = 0.139$.

Our findings with respect to the inheritance metrics suggest that both of these metrics should be excluded from the multivariate model. Given that RFC and LOC, RFC and NMC, and NMC and LOC are highly correlated we also exclude NMC from the multivariate regression model since this metric explains considerably less of the variation in the number of issues in a class compared to the other two metrics.

Table 15 presents the results of this analysis. The multivariate model with LOC, RFC and CBO as independent variables can predict 69% of the variation (compared to 43% in [32]) in the dependent variable (number of issues per class). Our multivariate linear regression model is only slightly better than the univariate linear model with either LOC or RFC as the independent variable.

3.5 Validity

The construct, internal and external validity of this empirical study is considered in this section.

3.5.1 Construct Validity

Construct validity focuses on how accurately the metrics utilised measure the phenomena of interest. With respect to the software metrics extracted from the source code, a commercially available tool that has been widely adopted in both academia and industry was used for this purpose. This tool calculated the Chidamber and Kemerer metrics which we utilised in this study. However, there has been considerable discussion in the literature with respect to how some of these metrics should be implemented, especially CBO. We utilised the definitions provided by Understand for Java. In this way our study can be replicated easily. The other two metrics utilised (number of changes and number of faults/issues) are dependent on the accuracy of Venus, the tool we used to link CVS and Bugzilla information. Eclipse provides considerable support for this process and therefore we are confident that these metrics are extracted accurately. However, as has been noted earlier, the number of faults/issues reflects all issues that are reported in Bugzilla and so may contain issues that would not be categorised as corrective maintenance and therefore should not be classified as faults. Having said that a majority of the issues were classified as normal severity, with a smaller proportion classified as enhancements. However the severity and priority of issues needs to be considered in more depth in the future.

3.5.2 Internal Validity

Internal Validity is the degree to which a causal relationship can be established between the independent variables and the dependent variables. From our results the best logistic regression for identifying classes with issues/faults was the multivariate regression. This model explained 37% of the variation in the dependent variable (number of issues/faults). However, this is considered a good result for logistic regression. When we changed the dependent variable to consider classes with more than ten faults the amount of variation explained by the model increased to 50.2%. The multivariate linear regression with number of issues/faults as the dependent variable explained 69% of the variation in the dependent variable. This is a reasonable result, however other metrics which may explain more of the variation in the dependent variable need to be identified and included in the model.

3.6 External Validity

The degree to which the findings from a study can be generalised to the whole population of interest constitutes the external validity. The JDT Core is of a considerable size and is comparable to many open-source projects [26]. However, while JDT Core may be of a similar size to many other open-source projects, further replication is required to confidently propose that the findings identified in this paper hold across many projects in the open-source domain. In addition, it may not be possible to extend the findings of studies involving open-source software systems to proprietary software due to the different development practices adopted [39, 43]. However it is also now claimed that many open-source projects have many similar characteristics to proprietary systems [47]. Further validations with both open-source and proprietary software systems are necessary to help us draw stronger conclusions and help identify the causal mechanisms for external quality attributes.

4. CONCLUSIONS

The empirical study reported in this paper has identified a number of interesting findings in relation to the nature of changes in an open-source software system. Most of the changes applied to a system are changes to existing methods or classes, with a smaller proportion relating to addition and deletion of methods. It was surprising that most issues are reported with normal severity. This suggests that reporters abide by the guidelines for creating issue reports.

Our findings confirm earlier results that Pareto's Law holds for the changes applied to the system. In relation to predicting classes associated with issues/faults a number of general conclusions can be drawn. First the inheritance based metrics NOC and DIT were not very useful in any of the

prediction models. This is probably due to the fact that the metrics themselves do not show much variation and thus are not good for the purpose of distinguishing different classes. The coupling metrics CBO and RFC along with LOC were the best predictors of fault-prone classes supporting much of the empirical work presented in Table 1. While our results suggest that these metrics are better predictors than the results presented by Gyimothy *et al.* these differences may be due to variations in the systems analysed. Variations in the systems should be studied further to try to identify which metrics sets are most suitable to which systems.

Future research will investigate the evolution of open source software systems in more detail addressing a number of specific areas. These include building models to identify the most fault-prone classes in systems and building prediction models for issues with different severity and priority levels. Given that previous findings suggest that the Chidamber and Kemerer metrics can predict classes with low severity faults better than classes with high severity faults [57] suggests that this is an area which requires further investigation.

5. ACKNOWLEDGEMENTS

This work was supported, in part, by Science Foundation Ireland grant to Lero - the Irish Software Engineering Research Centre (www.lero.ie). I would also like to acknowledge the support of the CSIS Department, University of Limerick.

6. REFERENCES

[1] F. Abreu and R. Capapuça. Object-oriented software engineering: measuring and controlling the development process. In *Fourth International Conference on Software Quality, ASQC*, McLean, VA, USA, October 1994.

[2] F. Abreu and R. Carapuça. Candidate metrics for object-oriented software within a taxonomy framework. *Journal of Systems and Software*, 26(1):87–96, 1994.

[3] F. Abreu and W. Melo. Evaluating the impact of object-oriented design on software quality. In *Proceedings of the Third International Software Metrics Symposium*, pages 90–99, 1996.

[4] K. Ayari, P. Meshkinfam, G. Antoniol, and M. Di Penta. Threats on building models from cvs and bugzilla repositories: the mozilla case study. In *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 215–228, New York, NY, USA, 2007. ACM.

[5] R. Bandi, V. Vaishnavi, and D. Turk. Predicting maintenance performance using object-oriented design complexity metrics. *IEEE Transactions on Software Engineering*, 29(1):77–87, 2003.

[6] V. Basili, L. Briand, and W. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, 1996.

[7] M. Bocco, D. Moody, and M. Piattini. Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation. *Journal of Software Maintenance and Evolution*, 17(3):225–246, 2005.

[8] L. Briand, J. Daly, V. Porter, and J. Wüst. A comprehensive empirical validation of design measures for object-oriented systems. In *Fifth International Symposium on Software Metrics*, pages 246–257, November 1998.

[9] L. Briand, J. Daly, and J. Wüst. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering*, 25(1):91–121, January/February 1999.

[10] L. Briand, P. Devanbu, and W. Melo. An investigation into coupling measures for C++. In *International Conference on Software Engineering*, pages 412–421, 1997.

[11] L. Briand, J. Wüst, J. Daly, and D. Porter. Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of Systems and Software*, 51(3):245–273, 2000.

[12] L. Briand, J. Wüst, S. Ikonomovski, and H. Lounis. Investigating quality factors in object-oriented designs: An industrial case study. In *International Conference on Software Engineering*, pages 345–354, 1999.

[13] A. Brooks, J. Daly, J. Miller, M. Roper, and M. Wood. Replication of experimental results in software engineering. Technical Report 96-10, ISERN, Livingstone Tower, Richmond Street, Glasgow G1 1XH, UK, 1996.

[14] M. Cartwright. An empirical view of inheritance. *Information and Software Technology*, 40(14):795–799, 1998.

[15] A. Chen, E. Chou, J. Wong, A. Y. Yao, Q. Zhang, S. Zhang, and A. Michail. Cvssearch: Searching through source code using cvs comments. In *In IEEE International Conference Software Maintenance (ICSM 2001*, pages 364–374, 2001.

[16] S. Chidamber and C. Kemerer. A metrics suite for object-oriented design. *IEEE Trans. Software Eng.*, 20(6):476–493, 1994.

[17] B. Cleary and C. Exton. The cognitive assignment eclipse plug-in. In *Proceedings of the 14th IEEE International Conference on Program Comprehension*, 2006.

[18] S. Counsell and P. Newson. Use of friends in C++ software: An empirical investigation. *Journal of Systems and Software*, 53(1):15–21, 2000.

[19] D. Cubranic and G. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 25th International Conference on Software Engineering*, pages 408–418, Portland, Oregon, May 2003.

[20] T. DeMarco. *Controlling Software Projects*. Yourdon Press Inc., New York, 1978.

[21] R. Dromey. A model of software product quality. *IEEE Transactions on Software Engineering*, 21(2):146–162, February 1995.

[22] K. El Emam, S. Benlarbi, N. Goel, and S. Rai. A validation of object-oriented metrics. Technical report, National Research Council of Canada, 1999.

[23] K. El Emam, S. Benlarbi, N. Goel, and S. Rai. The confounding effect of class size on the validity of object-oriented metrics. *IEEE Transactions on*

Software Engineering, 27(7):630–650, 2001.

[24] K. El Emam, W. Melo, and J. Machado. The prediction of faulty classes using object-oriented design metrics. *Journal of Systems and Software*, 56(1):63–75, 2001.

[25] M. English. Examining the structural features of systems developed in c++ and java. In *submitted to PPIG 09*, 2009.

[26] M. English, J. Buckley, T.Cahill, and K. Lynch. Measuring the impact of friends on the internal attributes of software systems. In *Fifth International Workshop on Source Code Analysis and Manipulation*, pages 151–160, September 2005.

[27] N. Fenton, M. Neil, W. Marsh, P. Hearty, L. Radlinski, and P. Krause. Project data incorporating qualitative factors for improved software defect prediction. In *International Conference on Predictor Models in Software Engineering (Promise)*, 2007.

[28] N. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering*, 26(8):797–814, August 2000.

[29] F. Fioravanti and P. Nesi. A study on fault-proneness detection of object-oriented systems. In *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, pages 121–130, 2001.

[30] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Proceedings of the International Conference on Software Maintenance*, Amsterdam, Netherlands, Sept. 2003.

[31] D. Glasberg, K. El Emam, W. Melo, and N. Madhavji. Validating object-oriented design metrics on a commercial java application. Technical report, National Research Council of Canada, 2000.

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

[33] R. Harrison, S. Counsell, and R. Nithi. An overview of object-oriented design metrics. In *International Conference on Software Technology and Engineering Practice,(STEP)*, pages 230–234. IEEE Computer Society Press, July 1997.

[34] R. Harrison, S. Counsell, and R. Nithi. An investigation into the applicability and validity of object-oriented design metrics. *Empirical Software Engineering*, 3(3):255–273, 1998.

[35] T. E. Hastings and A. S. M. Sajeev. A vector-based approach to software size measurement and effort estimation. *IEEE Transactions on Software Engineering*, 27(4):337–350, 2001.

[36] ISO/IEC. *ISO/IEC 9126-1 Software Engineering -Product Quality- Part 1: Quality Model*, 2001.

[37] R. Jeffery and L. Votta. Guest editor's special section introduction. *IEEE Transactions on Software Engineering*, 25(4):435–437, 1999.

[38] Y. Jiang, B. Cukic, T. Menzies, and N. Bartlow. Comparing design and code metrics for software quality prediction. In *International Conference on Predictor Models in Software Engineering*, 2008.

[39] K. Johnson. A descriptive process model for open-source software development. Master's thesis, University of Calgary, 2001.

[40] A. G. Koru and H. Liu. Identifying and characterizing change-prone classes in two large-scale open-source projects. *Journal of Systems and Software*, 80(1):63–73, January 2007.

[41] K. Krippendorff. *Notes from Content Analysis - an introduction to its methodology*. Sage Publications, 1980.

[42] M. Lorenz and J. Kidd. *Object-Oriented Software Metrics*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1994.

[43] A. MacCormack, J. Rusnak, and C. Baldwin. Exploring the structure of software designs: an empirical study of open source and proprietary code. *Management Science*, 52(7):1015–1030, July 2006.

[44] J. McCall, P. Richards, and G. Walters. Factors in software quality. Technical Report Ad/A-049-014/015/055, National Technological Information Service, 1977.

[45] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007.

[46] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *International Conference in Software Engineering*, pages 452–461, 2006.

[47] J. Noll. *Open Source Development Communities and Quality*, volume 275/2008, chapter Requirements Acquisition in Open Source Development:Firefox 2.0, pages 69–79. Springer Boston, 2008.

[48] T. Ostrand, E. Weyuker, and R. Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, 31(4):340–355, April 2005.

[49] D. Perry, A. Porter, and L. Votta. Empirical studies of software engineering: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 345–355, New York, NY, USA, 2000. ACM Press.

[50] Scientific Toolworks Inc. Scitools maintenance, metrics and documentation tools for Ada, C, C++, Java and Fortran. web: www.scitools.com, 2006.

[51] J. Sliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *Proceedings of the International Workshop on Mining Software Repositories*, pages 1–5, St. Louis, Missouri, May 2005.

[52] R. Subramanyam and M. Krishnan. Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on Software Engineering*, 29(4):297–310, 2003.

[53] M. Tang, M. Kao, and M. Chen. An empirical study on object-oriented metrics. In *Proceedings of the Sixth International Symposium on Software Metrics*, pages 242–249, 1999.

[54] The Eclipse Foundation. Eclipse bugzilla home. https://bugs.eclipse.org/bugs/. accessed 05/02/08.

[55] M. Ware, F. Wilkie, and M. Shapcott. The application of product measures in directing software maintenance

activity. *Journal of Software Maintenance and Evolution: Research and Practice*, 19:133–154, 2007.

[56] P. Yu, T. Systa;, and H. Muller. Predicting fault-proneness using oo metrics: An industrial case study. In *CSMR '02: Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, pages 99–107, Washington, DC, USA, 2002. IEEE Computer Society.

[57] Y. Zhou. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Trans. Softw. Eng.*, 32(10):771–789, 2006.

[58] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects in eclipse. In *International Conference on Precictor Models in Software Engineering (Promise)*, 2007.