SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

December 2016, Vol. 59 122901:1–122901:18
doi: 10.1007/s11432-015-5426-3

Empirical analysis of network measures for predicting high severity software faults

Lin CHEN[1], Wanwangying MA[1], Yuming ZHOU[1], Lei XU[1], Ziyuan WANG[2], Zhifei CHEN[1] & Baowen XU[1*]

[1]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210023*, China;*
[2] *School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing* 210023*, China*

Received May 18, 2016; accepted June 18, 2016; published online November 7, 2016

Abstract Network measures are useful for predicting fault-prone modules. However, existing work has not distinguished faults according to their severity. In practice, high severity faults cause serious problems and require further attention. In this study, we explored the utility of network measures in high severity fault-proneness prediction. We constructed software source code networks for four open-source projects by extracting the dependencies between modules. We then used univariate logistic regression to investigate the associations between each network measure and fault-proneness at a high severity level. We built multivariate prediction models to examine their explanatory ability for fault-proneness, as well as evaluated their predictive effectiveness compared to code metrics under forward-release and cross-project predictions. The results revealed the following: (1) most network measures are significantly related to high severity fault-proneness; (2) network measures generally have comparable explanatory abilities and predictive powers to those of code metrics; and (3) network measures are very unstable for cross-project predictions. These results indicate that network measures are of practical value in high severity fault-proneness prediction.

Keywords network measures, high severity, fault-proneness, fault prediction, software metrics

Citation Chen L, Ma W W Y, Zhou Y M, et al. Empirical analysis of network measures for predicting high severity software faults. Sci China Inf Sci, 2016, 59(12): 122901, doi: 10.1007/s11432-015-5426-3

1 Introduction

As software systems become larger and increasingly complex, it is becoming more difficult and costly to test source code and to produce high-quality products. In practice, testing resources are often limited. Therefore, it is important to allocate testing resources to higher risk modules that may contain severe faults before delivery. Many prediction models have been proposed to prioritize modules in terms of the number of potential faults. However, few studies have considered the severity of faults [1,2], which defines the impact that a given fault has on software.

Severe faults can result in major function loss, extensive data corruption, and even termination of the entire system. During software development, leaving a module with many undetected trivial bugs will

* Corresponding author (email: bwxu@nju.edu.cn)

not seriously affect the quality. In contrast, leaving a module with a few undetected severe faults can lead to disaster. Thus, classes with highly severe faults should be tested and fixed before classes with low severity faults. Categorizing faults according to different severity levels helps prioritize the sequence in which faults are fixed. A high severity fault prediction model would help software engineers focus testing resources on the most severe faults to reduce and assess the most influential failures and improve quality prior to deploying the software.

In the literature, various metrics to characterize software entities and build prediction models have been reported, such as code metrics (CMs) [3–6], process metrics [7–9], and previous defects [10, 11]. More recently, network measures derived using concepts from the social network analysis field [12] have been employed in fault-proneness prediction [13–19]. Network-based analysis treats modules as nodes and extracts the dependencies between them as edges to construct software source code network. Then, the obtained network measures are used to build prediction models. By considering the interactions between modules, network measures characterize the information flow and overall topology of a software system, which cannot be captured by code metrics. However, existing work on network measures has not examined their fault severity predictive capabilities. In this study, we investigated the actual usefulness of network measures for predicting high severity faults.

We conducted our study using multiple releases of four projects: Mozilla Firefox, Eclipse, Apache Ant, and Apache Hbase. By extracting the data and call dependencies between modules (files or classes), source code networks were constructed at the module level. We then empirically investigated most network measures proposed in the literature to determine how they characterized high severity fault-proneness. We also evaluated their predictive effectiveness compared to that of traditional code metrics in the context of forward-release and cross-project prediction. Specifically, we aimed to answer the following research questions:

• **RQ1: Is there a significant association between each network measure and high severity fault-proneness?** To analyze the effect of each network measure on fault-proneness, the purpose of RQ1 was to determine whether each of the network measures is a potentially useful predictor for high severity fault-proneness.

• **RQ2: How effective is the explanatory ability of network measures for high severity fault-proneness?** After analyzing each network measure, we further investigated the effects of the network measures when used together. By considering the correlation between network measures, RQ2 selected metrics to construct models that could optimize the explanation of high severity fault-proneness and evaluate their explanatory abilities.

• **RQ3: How effective is the predictive power of network measures for high severity fault-proneness?** The models obtained in RQ2 were used to predict the high severity fault-proneness of modules. In industrial use, historical project data are often exploited to estimate the quality of a newly released version of the same project. The association between network measures and high severity fault-proneness of past releases can be used to predict high severity faults in the next release. For RQ3, we evaluated the forward-release predictive power of network measures.

• **RQ4: How effective is the portability of network measures for high severity fault-proneness?** In addition to intra-project prediction, we were also interested in how well network measures perform in cross-project fault prediction since it is useful to transfer prediction models of other projects to projects without historical data. For RQ4, we investigated the portability of network measures across different projects.

This paper makes a number of contributions. First, we validated the association between each network measure and high severity fault-proneness. Second, we compared the effectiveness of network measures in high severity fault-proneness prediction to four categories of the most commonly used code metrics including size, structural complexity, Halstead's software science metrics, and object-oriented design metrics. Third, most publicly available fault data sets, including Promise [1)] , do not contain information about fault severity, possibly because it is time consuming and expensive to collect qualitative data [20].

1) http://openscience.us/repo/.

Table 1 The statistics of the studied projects

Project	#releases	Language	#modules		kLOC		#faults	
			First release	Last release	First release	Last release	All	High
Firefox	6	C/C++	11431	12721	2613	2881	4490	752
Eclipse	5	Java	6014	12936	815	1979	13073	2072
Ant	4	Java	412	1053	41	116	833	137
Hbase	8	Java	513	1889	104	571	2861	538

Therefore, our experimental data [2)] are a valuable supplement to such data sets.

The remainder of this paper is organized as follows. Section 2 describes the design of our study, including the research questions, data sources, method used to collect experimental data sets, construction of source code networks, and network measures together with the most commonly used code metrics. Section 3 introduces the dependent and independent variables, presents the employed modeling technique, and describes our data analysis methods. The experimental results and the answers to our research questions are reported in Section 4. Section 5 discusses the results and threats to the validity of our study. Section 6 provides related work. Section 7 concludes the paper and outlines directions for future work.

2 Study design

In this section, we introduce the subject projects investigated in our study, the manner by which we collected and processed fault data, the construction of source code networks, the network measures and the code metrics we employed.

2.1 Studied projects

We performed our study using four well-known open source projects: Firefox [3)], Eclipse [4)], Ant [5)], and Hbase [6)]. The projects vary in their application domains, size, and programming languages. Firefox is a popular web browser developed by the Mozilla Foundation. Eclipse is a well-known integrated development environment from the Eclipse Foundation. The last two projects are parts of the Apache Software Foundation. Specifically, Ant is a software tool for automating software build processes, and Hbase is a non-relational, distributed database. With respect to software size, Firefox and Eclipse are large-scale projects, while the two Apache projects are relatively small. Apart from Firefox which is primarily written in C/C++, the other projects are all Java applications.

Table 1 lists some properties of these software systems. The second column shows the number of releases we analyzed. The third column presents the main programming language(s) used in each project. The columns entitled #modules and kLOC indicate the project size in terms of the total number of files (for C/C++ projects)/classes (for Java projects) and total number of lines of code, respectively.

2.2 Fault data collection and processing

In this study, we investigated the predictive power of network metrics with regard to fault severity. Thus, we were interested in the fault-proneness of each module at high severity level. To collect fault data for these projects, we first extracted the change logs with key words (e.g., "bug" for Firefox) that were found in commits from the version control system (GIT [7)] or SVN [8)]). Then, the bug reports with

2) https://github.com/xiaobena/network-measures/tree/master/with%20severity.
3) ttp://www.firefox.com/.
4) https://www.eclipse.org/.
5) http://ant.apache.org/.
6) http://hbase.apache.org/.
7) http://git-scm.com/.
8) http://subversion.apache.org/.

corresponding bug IDs were obtained from the bug tracking system (Bugzilla [9] or Jira [10]). We only retained the faults for which the Resolution field was set to "FIXED" or the Status field was set to "CLOSED". We parsed the change logs and bug reports to obtain the following bug information: bug ID, severity (for Bugzilla)/priority (for Jira), and a list of files modified to fix the faults.

In Bugzilla, there are seven levels in the Severity field: blocker, critical, major, normal, minor, trivial, and enhancement (from most to least severe). We excluded bug reports with the enhancement severity because they are not real bugs. Faults labeled major, critical or blocker are classified as high severity. In Jira, the Priority field indicates the severity of faults. Five levels of priority are defined in Jira: blocker, critical, major, minor, and trivial (from most to least severe). Faults labeled blocker or critical are classified as high severity.

In addition, when collecting fault data, we did not distinguish between security bugs and ordinary bugs. The severity of each security bug was identified based on the severity or priority field of its bug report. However, we are not authorized to access some of the latest security bugs; thus, they are excluded from our study.

The last four columns of Table 1 show the total number of faults and the number of high severity faults collected for the four projects. As can be seen, Eclipse has the most high severity faults, while Ant has the fewest. For each project, high severity faults account for 15.8%–18.8% of all faults.

2.3 Construction of the source code network

We generated the source code network at the module level and considered a single file (for C/C++ projects) or class (for Java projects) as a module. Each module was considered a single node in the network, and the dependencies between modules were extracted as edges. A program dependence is a direct relationship between two pieces of code [13]. We collected two types of dependencies, namely, data dependence between the use of a variable and its definition and call dependence from the sites where a function is called to its declaration. The two types of dependencies were rolled up to the module level. Note that Firefox is C/C++ software. We combined each .cpp/.c file with the corresponding header file (.h) as a single node and merged their dependencies. Even though self-dependencies exist, we did not consider them in this study.

To construct the source code network, we used the Understand [11] program-understanding tool. For each project version, we first generated an Understand database to store information about the entities (e.g., files and classes) and references (e.g., function calls and variable references). Then a Perl script was used to track the information dependencies between modules and to generate the software network at the module level.

2.4 Network measures

Each node in the source code network was evaluated from two perspectives, namely, local and global. To view a given node locally, we considered an ego network, which consists of the node itself and every other node that connects to it directly. Ego network measures (ENs) capture the importance of the node within its neighborhood. A global network contains all nodes, and global network measures (GNs) weight the importance of the nodes within the entire network. For each module in the software system, the ego network only considers the direct dependencies between modules, while the global network considers the indirect uses of data and methods.

We employed Ucinet [12] to obtain the ENs and GNs. Table 2 summarizes these network measures.

For each node, we computed three types of ego network: in (_in), out (_out), and undirected (_un). The in (out) ego network only considers incoming (outgoing) directed dependencies between an ego module and its neighboring modules, while the undirected ego network contains both types of dependencies.

9) https://bugzilla.mozilla.org/, https://bugs.eclipse.org/bugs/.
10) https://issues.apache.org/jira/secure/Dashboard.jspa.
11) http://www.scitools.com/.
12) https://sites.google.com/site/ucinetsoftware/home/.

Table 2 The network measures in this study

Category	Metric	Description
Ego network measures	Size	# nodes that ego is directly connected to
	Ties	# directed ties corresponds to the number of edges
	Pairs	# unique pairs of nodes, i.e., Size × (Size − 1)
	Density	% of possible ties that are actually present, i.e., Ties/Pairs
	nWeakComp	# weak components in the ego network
	pWeakComp	# weak components normalized by size
	2StepReach	# nodes ego can reach within two steps normalized by Size
	ReachEffic	2StepReach normalized by the sum of the size of the ego's every neighbor's ego network
	Broker	# pairs not directly connected to each other
	nBroker	Broker normalized by the number of pairs
	EgoBetween	% shortest paths between neighbors that pass through ego
Global network measures	Degree	# nodes adjacent to a given node
	Clus Coef	measures the density of an node's open neighborhood
	Closeness	sum of the lengths of the shortest paths from a node to all other nodes
	Reachability	# nodes that can be reached from a given node
	Eigenvector	assigns relative scores to all nodes in the dependency graphs
	Betweenness	measures how many shortest paths between other entities it occurs
	Power	measures the connections of the nodes in one's neighborhood
	EffSize	# nodes that are connected to a node minus the average number of ties between these nodes
	Efficiency	normalizes EffiSize to the total size of the network
	Constraint	measures how strongly an node is constrained
	Hierarchy	measures the extent to which constraint a node is concentrated in the network

In addition, some global metrics were computed using only the incoming or outgoing edges, which are indicated as IN and OUT respectively. In total, we explored the usefulness of 53 network measures (36 ENs and 17 GNs) in high severity fault-proneness prediction.

2.5 Code metrics

In this study, code metrics are used to provide a performance baseline for our prediction models. We applied three types of widely used code metrics: size metrics, structural complexity metrics, and Halstead metrics [21]. A set of object-oriented metrics given by Chidamber and Kemerer [22] (CK metrics) including CBO, RFC, LCOM, NOC, DIT, and WMC were applied to the Java projects. Thus, 17 and 22 code metrics were computed for all versions of the C/C++ projects and Java projects, respectively. Table 3 summarizes these code metrics.

We collected these code metrics using the above-mentioned Understand databases and a Perl script. The size and structural complexity metrics were calculated at the module (file or class) level, while the Halstead metrics were generated for each single function and rolled up to the module level. Since Firefox is primarily written in the C language, and most of its files contain no classes; thus, we did not compute CK metrics for these files. CK metrics were collected for classes in the three Java projects.

3 Analysis approach

In this section, we first introduce the dependent and independent variables investigated in our study. We then describe the research hypotheses corresponding to the four research questions, the employed modeling technique, and the data analysis methods.

Table 3 The baseline code metrics in this study

Category	Metric	Description
Size metrics	CL	Number of all lines of a module
	LOC	Number of lines containing source code of a module
	LOCE	Number of lines containing executable source code of a module
Structural complexity metrics	CCMax	Maximum cyclomatic complexity of all nested functions in a module
	CCSum	Sum of cyclomatic complexity of all nested functions in a module
	NMax	Maximum nesting level of control constructs in a module
Structural complexity metrics	$n1$	Total number of distinct operators of a function
	$n2$	Total number of distinct operands of a function
	n	Total number of vocabulary of a function, $n = n1 + n2$
	$N1$	Total number of operators of a function
	$N2$	Total number of operands of a function
	N	Halstead program length, $N = N1 + N2$
	V	Halstead program volume, $V = N \times \log_2 n$
	D	Halstead program difficulty, $D = n1/2 \times N2/n2$
	L	Halstead program level, $L = 1/D$
	E	Halstead programming effort, $E = V \times D$
	T	Programming time, $T = E/18$ s
Chidamber and Kemerer metrics	WMC	The number of methods implemented within a given class.
	DIT	The length of the longest path from a given class to the root in the inheritance hierarchy.
	NOC	The number of classes that directly inherit from a given class.
	CBO	The number of distinct non-inheritance-related classes on which a given class is coupled. A class is said to be coupled to another class if it uses methods or attributes of the other.
	RFC	The number of methods that can potentially be executed in response to a message being received by an object of a given class.
	LCOM	For each attribute in a given class, calculate the percentage of the methods in the class using that attribute.

3.1 Variable description

The goal of our study was to explore the relationship between network measures and high severity fault-proneness of modules. The binary dependent variable in this study is fault-proneness, whose definition must take into account the context in which a module is used. When investigating the fault-proneness prediction capabilities of the metrics in the context of high severity faults, a module was considered fault-prone if it had at least one high severity fault, while a module was not fault-prone if it had no high severity fault.

The independent variables in this study consisted of two sets, specifically, 53 network measures and 22/17 most commonly used code metrics. All of the metrics were collected at the module level. In this study, the classification of modules into fault-prone and not fault-prone was performed using logistic regression (described in Subsection 3.3). We compared models built with the ENs, the GNs, both types of network measures (NMs), the CMs, and all metrics (CNs). That is, for each version of the four projects, we built five prediction models.

3.2 Research hypotheses

The purpose of RQ1 was to investigate whether significant association between each network measure and high severity fault-proneness exists. Taking into consideration interactions between modules, each type of network measure captures the topology of a software system from a certain perspective, and evaluates the importance of each module relative to a given network characteristic. In addition, software structure is supposed to correlate with fault-proneness. Thus, we conject network measures, which

describe dimensions of the software structure, were related to fault-proneness. We put up a hypothesis testing for each single network measure for RQ1. A generic form of the null hypothesis $H1_0$ and its alternative hypothesis $H1_A$ are stated as follows.

• $H1_0$: There is no significant association between network measure x and fault-proneness at high severity level.

• $H1_A$: There is a significant association between network measure x and fault-proneness at high severity level.

The purpose of RQ2 was to analyze the explanatory ability of network measures for high fault-proneness. When used together, network measures correlate and describe the software structure from multiple perspectives. Here, code metrics were used as a baseline to evaluate the goodness of the explanatory ability of network measures. We formulated the following null hypothesis $H2_0$ and alternative hypothesis $H2_A$.

• $H2_0$: Network measures do not have better explanatory ability than code metrics for fault-proneness at a high severity level .

• $H2_A$: Network measures have a better explanatory ability than code metrics for fault-proneness at a high severity level.

The purpose of RQ3 was to evaluate the predictive power of network measures for high severity fault-proneness. There are no universal standards to determine whether predictive power is good; therefore, for RQ3, code metrics were used as a baseline. We formulated the following null hypothesis $H3_0$ and alternative hypothesis $H3_A$.

• $H3_0$: Network measures do not have a better predictive power for fault-proneness at high severity level than code metrics.

• $H3_A$: Network measures have a better predictive power for fault-proneness at high severity level than code metrics.

The purpose of RQ4 was to evaluate the portability of network measures for fault-proneness at a high severity level. Here, code metrics were still used to compare the portability of network measures. We formulated the following null hypothesis $H4_0$ and alternative hypothesis $H4_A$.

• $H4_0$: Network measures do not have a better portability than code metrics for fault-proneness at high severity level.

• $H4_A$: Network measures have a better portability than code metrics for fault-proneness at high severity level.

3.3 Modeling technique

Logistic regression was used to predict the dependent variable (fault-proneness) from a set of independent variables (network measures and code metrics) to determine the percent of the variance in the dependent variable that could be explained by the independent variables [3]. Logistic regression is a type of probabilistic statistical classification model in which the dependent variable can take two different values. Since modules in this study are divided into two categories, namely, fault-prone and not fault-prone, logistic regression is suitable for building the fault-proneness prediction models. There are two types of logistic regression, univariate and multivariate logistic regression. The univariate analysis was used to find the individual effect of the independent variable on the dependent variable, while the multivariate analysis was used to find the combined effect of the independent variables on the dependent variable.

Odds ratio is the most commonly used measure to quantify the magnitude of an association between independent and dependent variables in logistic regression models. When a logistic regression is calculated, the regression coefficient is the estimated increase in the log odds of the dependent variable per unit increase in the value of the independent variable [23]. An odds ratio 1 means that the independent variable does not affect the dependent variable. An odds ratio greater than 1 indicates that the independent variable positively associates with the dependent variable, while an odds ratio less than 1 indicates that the independent variable negatively associates with the dependent variable.

3.4 Data analysis methods

In this section, we describe the data analysis methods used to answer the research questions.

3.4.1 *Univariate logistic regression analysis for RQ*1

We employed univariate logistic regression to determine whether there was a significant association between each network measure and fault-proneness at a high severity level. Univariate regression analysis is used to examine the effect of each metric separately, identifying which metrics were significantly related to the fault-proneness of modules to determine which network measures are potentially useful high severity fault-proneness predictors.

Regression can be significantly affected by influential observations. When performing univariate analysis, we checked for the presence of influential observations using Cook's distance [24]. Cook's distance is a measure of how much the residuals of all observations would change if a particular observation were excluded from the calculation of the regression coefficients. A large Cook's distance indicates that excluding an observation from computation of the regression statistics would change the coefficients substantially. An observation with a Cook's distance equal to or greater than 1 is considered as an influential observation [24] and is thus excluded from analysis.

The following statistics were evaluated for each single network measure.

• Coefficient: The estimated regression coefficient that represents the change in the logit for each unit change in the independent variable. A larger absolute value of the coefficient indicates a stronger impact of the independent variable (a given network measure) on the possibility of a module being fault-prone.

• P-value: This statistic relates to the statistical hypothesis and indicates whether the corresponding coefficient is significant or not. In this study, the Wald test [25] was used to assess the significance of an individual metric. We set the significance level $\alpha = 0.05$ to assess the obtained p-value.

• Odds ratio: Similar to coefficient, this statistic quantifies the effect of a given network measure on fault-proneness and enables comparison of the effects of individual measures.

For each metric, the null hypothesis $H1_0$ corresponding to RQ1 was rejected if the p-value of univariate logistic regression was less than 0.05.

3.4.2 *Multivariate logistic regression analysis for RQ*2

To analyze the explanatory ability of network measures for fault-proneness at high severity level, we used multivariate logistic regression to construct regression models that could optimize the explanation of fault-proneness and analyze the effect of the network measures when used together. A multivariate logistic regression model was used to indicate how well the network measures interpreted fault-proneness.

To choose which network measures to include in the regression model, we adopted a backward stepwise regression variable selection technique, which involved starting with all of the candidate variables, testing the deletion of each variable using a chosen model comparison criterion, deleting the variable that improved the model the most by being deleted, and repeating this process until no further improvement was possible.

In addition, one common problem in logistic regression analysis is multicollinearity which refers to a situation in which two or more independent variables are related in a highly manner. In the presence of multicollinearity, the estimated regression coefficients may be inaccurate and unreliable, which can adversely affect the determination of how individual independent variables contribute to the understanding of the dependent variable. In this study, we computed the variance inflation factors (VIFs) of each metric to examine the multicollinearity between the variables of the model. We removed all variables with VIFs greater than 10, which is the recommended cut-off value when dealing with multicollinearity in a regression model [26].

Furthermore, all of the results were checked for the presence of influential observations using Cook's distance. Observations with Cook's distances equal to or greater than 1 were considered influential observations and were thus excluded from analysis.

The prediction models were built with five metric sets: ENs, GNs, NMs, CMs, and CNs. The construction of these models involved three main steps: (1) metric selection using backward stepwise regression; (2) eliminating metrics with VIF > 10; and (3) removing observations with Cook's distance $\geqslant 1$.

The goodness of fit of a statistical model describes how well it fits a set of observations and evaluates how well the independent variables explain the dependent variable. Thus, to evaluate the explanatory ability of network measures for fault-proneness at a high severity level, we only needed to analyze the goodness of the fit of the regression models built with network measure. The goodness of fit in linear regression models is generally measured using the R squared value. This value has no direct analog in logistic regression; therefore, the following statistics were used to evaluate the fit of each prediction model.

• -2-Log-likelihood: A measure based on deviance that represents the lack of fit to the data in a logistic regression model [26]. A smaller value indicates a better fit.

• Cox&Snell-R2: An alternative index of goodness of fit related to the R squared value from linear regression. It ranges from 0 to 0.75 [27].

• Nagelkerke-R2: A correction to the Cox and Snell R Squared value that is used to ensure that the maximum value is equal to 1 [28].

To answer RQ2, we compared the goodness of fits of regression models built with network measures and code metrics using Wilcoxon signed-rank tests. Note that we distinguished three sets of network measures: the ENs, the GNs, and the NMs. The Wilcoxon tests were employed to compare each of the above statistics obtained from the models built with three pairs of metric sets: (I) ENs vs. CMs; (II) GNs vs. CMs; and (III) NMs vs. CMs. In particular, we used Benjamini-Hochberg (BH) [29] corrected p-values to examine whether a difference is significant at the significance level of 0.05.

3.4.3 *Forward-release prediction for RQ3*

Forward-release prediction is considered more suitable and practical for industrial use when past project data is exploited to predict fault-prone modules in future releases. In our study, we evaluated the predictive power of network measures under this most common case where data from one release were trained to test the next immediate release. For example, we build the logistic regression model with release 0.92 of Hbase to test against release 0.94 of the same project.

Note that our goal was to classify each module of the next release into one of two categories, namely, fault-prone or not fault-prone, while logistic regression model were used to generate a predicted probability surface. Therefore, the choice of a threshold above which a given module would be predicted to be fault-prone was required. The traditional default is to simply use a threshold of 0.5 as the cut-off; however, this threshold does not necessarily result in the highest prediction accuracy, especially for highly unbalanced data sets [30]. Thus, we employed a receiver operating characteristic (ROC) curve to determine the classification threshold.

An ROC curve graphically illustrates the performance of a binary classifier at various discrimination thresholds, with the true positive rate (TPR = fraction of true positives out of the total actual positives) as the vertical axis and false positive rate (FPR = fraction of false positives out of the total actual negatives) as the horizontal axis. A good model will achieve a high TPR with a relatively small FPR. Thus, an ROC plot will rise steeply at the origin; and level off at a value near the maximum of 1. As a model with TPR = 1 and FPR = 0 can be considered ideal, the threshold that minimizes the distance between the point of the ROC curve and the point (0, 1) can be used as an optimization criteria [30].

For forward-release prediction, we first used each regression model obtained for RQ2 to run self-prediction on the training set to obtain the probabilities of fault-proneness, which were then used to obtain the classification threshold according to the ROC curve. These models and corresponding thresholds were employed to predict the fault-proneness of the modules in the next release. In addition, we adopted the following indicators to evaluate the prediction accuracy.

• Sensitivity: The percentage of modules correctly predicted to be fault-prone. According to the confusion matrix (Table 4), Sensitivity can be calculated as follows: Sensitivity $= tp/(tp + fn)$. The

Table 4 Confusion matrix

		Acutal	
		Fault-prone	Not fault-prone
Predicted	Fault-prone	tp: true positives	fp: false positives
	Not fault-prone	fn: false negatives	tn: true negatives

value of Sensitivity ranges from 0 to 1; higher value indicates better performance.

• Specificity: The percentage of modules correctly predicted as fault-prone. According to the confusion matrix, Specificity can be calculated as follows: Specificity $= tn/(tn+fp)$. The value of Specificity ranges from 0 to 1; higher value indicates better performance.

• Area under the ROC curve (AUC): The AUC is suitable for evaluating the performance of a classification model with unbalanced data sets. Unlike Sensitivity and Specificity, AUC is a threshold-independent accuracy measure for a binary classifier. A random classification has an AUC value of 0.5 while the perfect one has an AUC of 1.0, which signifies that the classifier correctly predicts the fault-proneness of all the modules. Therefore, higher AUC values indicate better classification model performance.

The above three indicators were used to represent the predictive powers of the network measures. As in RQ2, we used the Wilcoxon signed-rank tests to compare Sensitivity, Specificity, and AUC using three pairs of metric sets: (I) ENs vs. CMs; (II) GNs vs. CMs; and (III) NMs vs. CMs. In particular, we used BH corrected p-values to examine whether a difference was significant at a significance level of 0.05.

3.4.4 *Cross-project prediction for RQ*4

To analyze the portability of the network measures, we transferred the models built for one project to another and calculated the possibility of fault-proneness for each module in the new project.

Our goal was to classify each module of the new project into one of two categories, namely, fault-prone or not fault-prone; thus, the thresholds determined for RQ3 were used for RQ4. In other words, the models obtained for RQ2 and their corresponding thresholds were employed to predict the fault-proneness of modules in other projects. In addition, to evaluate the prediction accuracy, we adopted the three commonly used indicators: Sensitivity, Specificity and AUC. The Wilcoxon signed-rank tests were used to compare each of them using three pairs of metric sets: (I) ENs vs. CMs; (II) GNs vs. CMs; and (III) NMs vs. CMs. In particular, we use BH corrected p-values to examine whether a difference was significant at a significance level of 0.05.

4 Results

This section presents the results of our empirical study with respect to each research question.

4.1 Association (RQ1)

Tables 5 and 6 summarize the numbers of projects for which individual network measure has significantly positive or negative effect on high severity fault-proneness in at least 50% releases. A total number of 53 tests were run on each version of the four studied projects because there are 53 network measures. Since multiple tests on the same data set may result in spurious statistically significant results, we use the BH corrected p-values to control false discovery.

From the results, we see that most of the investigated network measures have significant correlations with fault-proneness at a high severity level. In addition, most network measures are positively related to fault-proneness because their estimated regression coefficients are positive. Thus, larger values of these network measures for a module indicate increased likelihood that the corresponding module is prone to high severity faults. However, only a few odds ratios are markedly greater than 1, specifically, nBroker, nBroker_in, nBroker_out, InCloseness, and Efficiency, which indicates that only a few network measures make outstanding contributions to the estimated probability of fault-proneness.

Table 5 The number of projects for which individual network has significantly positive effects on fault-proneness in at least 50% releases

Number of projects	Network measures
4	Size, Ties, 2StepReach, nBroker, 2StepReach_in, Size_out, Ties_out, Pairs_out, 2StepReach_out, Broker_out, nBroker_out, nEgoBetween_out, Degree, OutDegree, InCloseness, OutdwReach, IndwReach, Betweenness, EffSize
3	EgoBetween, nEgoBetween_in, n_Broker_in, nWeakComp_out, EgoBetween_out, InDegree, OutCloseness, Eigenvec, Indirects
2	Pairs, Broker, Size_in, Ties_in, nWeakComp_in, ReachEffic_in, Density_out, UnBetweenness, Efficiency
1	Density, nWeakComp, pWeakComp_in, nEgoBetween, Pairs_in, Density_in, Broker_in, EgoBetween_in

Table 6 The number of projects for which individual network has significantly negative effect on fault-proneness in at least 50% releases

Number of projects	Network measures
4	ClusCoef, Constraint
3	pWeakComp, pWeakComp_out, ReachEffic_out
2	ReachEffic, Power, Hierarchy
1	Density, Size_in, pWeakComp_in, ReachEffic_in, nBroker_in, Eigenvec, Indirects

Table 7 Results of Wilcoxon tests

Project	-2-Log-likelihood			Cox&Snell-R2			Nagelkerke-R2		
	ENs	GNs	EMs	ENs	GNs	NMs	ENs	GNs	NMs
Firefox	√	√	√	√	√	√	√	√	√
Eclipse	√	–	–	√	–	√	–	–	√
Hbase	√	√	√	–	√	√	–	–	√
Ant	–	–	–	–	–	–	–	–	–

Response to hypothesis $H1_0$: "There is no significant correlation between network measure x and fault-proneness at a high severity level." Partially reject. For most network measures, but not all, our univariate logistic regression analysis results, obtained from 23 data sets of four projects, reject the null hypothesis $H1_0$ for RQ1.

4.2 Explanatory ability (RQ2)

We used the multiple logistic regression analysis described in Subsection 3.4.2 to answer RQ2. Here, we will introduce and describe the prediction model for high severity faults, namely, the high severity fault prediction model (HSF model). We compared the HSF model built with the ENs, the GNs, the NMs, the CMs, and the CNs.

We first computed -2-Log-likelihood, Cox&Snell-R2 and Nagelkerke-R2 for each model with the CMs, ENs, GNs, NMs, and CNs for Firefox, Eclipse, Hbase, and Ant. We then used the Wilcoxon signed-rank sum tests to evaluate the differences between three compared pairs of metric sets: (I) ENs vs. CMs; (II) GNs vs. CMs; and (III) NMs vs. CMs at a significant level of 0.05.

Table 7 lists the outcomes of the Wilcoxon tests for high severity faults. The columns entitled ENs, GNs, and NMs show the results of comparing the ENs, GNs, and NMs, respectively, with the CMs. Here, √ indicates refusal of the null hypothesis, while a "–" indicates that the null hypothesis cannot be rejected. With respect to -2-Log-likelihood, Cox&Snell-R2, and Nagelkerke-R2, the p-values of the Wilcoxon tests show that the differences between each of the three compared pairs of metric sets are significant for Firefox, but not significant for Ant. However, for Eclipse and Hbase, we could not obtain consistent comparison results for the three statistics.

Since -2-Log-likelihood, Cox&Snell-R2, and Nagelkerke-R2 are indicators of the goodness of fit of the regression models, which in turn indicate the explanatory power of the independent variables (metric

sets) for the dependent variable (fault-proneness) at a high severity level, the results show that the ENs, GNs, and NMs are better than the CMs at interpreting fault-proneness for only some projects.

Response to hypothesis $H2_0$: "Network measures do not have a better explanatory ability than CMs for fault-proneness at a high severity level." Partially reject. Network measures have better explanatory ability in Firefox. However, the Wilcoxon test results are mixed for the other projects.

4.3 Predictive power (RQ3)

To answer RQ3, we used the multivariable logistic regression models built in Subsection 4.2 to predict the fault-proneness of modules at high severity level in the next release of the same project. Here Sensitivity, Specificity, and AUC were employed to evaluate the predictive effectiveness of these models. In addition, the Wilcoxon signed-rank test was used to compare the predictive power of three pairs of metric sets: (I) ENs vs. CMs; (II) GNs vs. CMs; and (III) NMs vs. CMs.

Figure 1 shows the distributions of Sensitivity, Specificity, and AUC obtained from the forward-release predictions for Firefox, Eclipse, Hbase, and Ant. Each box in the boxplot corresponds to the performance of an HSF model built with one of the five metric sets: CMs, ENs, GNs, NMs, and CNs.

For Firefox, the ENs, GNs, and NMs all have lower mean/median Sensitivity, and AUC value but higher mean/median Specificity values than the CMs. However, the results of the Wilcoxon tests indicate that only the Specificity of the GNs is significantly higher than that of the CMs.

For Eclipse, the ENs have higher mean/median Sensitivity values but lower mean/median Specificity values than the CMs have. The GNs and NMs both have higher mean/median Specificity values but lower mean/median Sensitivity values than the CMs have. The mean/median AUC values of the ENs, GNs, and NMs are all greater than those of the CMs. However, the p-values in the Wilcoxon tests are not significant for the three compared pairs of metric sets.

For Hbase, the ENs have lower mean/median Sensitivity, Specificity, and AUC values than the CMs. The GNs have higher mean/median Sensitivity and AUC values but lower Specificity values than the CMs have. The NMs have lower mean/median Specificity and AUC values but higher Sensitivity values than the CMs have. However, the results of the Wilcoxon tests do not show significant differences in the three compared pairs of metric sets.

For Ant, the ENs, GNs, and NMs all have lower mean/median Sensitivity, Specificity, and AUC values than the CMs have. Again, their differences are not statistically significant according to the BH corrected p-values.

Response to hypothesis $H3_0$: "Network measures do not have better predictive power than code metrics for fault-proneness at a high severity level". Partially reject. Combining the results for the four studied projects, we found that most p-values obtained from the Wilcoxon tests were not significant. That is, we cannot prove that network measures are more effective for predicting high severity faults than code metrics are. More specifically, in most cases, the predictive powers of the network measures were equal to that of the code metrics.

4.4 Portability (RQ4)

To answer RQ4, we employed the multivariable logistic regression models (Subsection 4.2) to predict fault-proneness at a high severity level across different projects. For each of the five metric sets (CMs, ENs, GNs, NMs, and CNs), we selected the HSF model built with the earliest version of one project to predict fault-prone modules in all studied versions of the other projects. Sensitivity, Specificity, and AUC were used to evaluate the cross-project predictive effectiveness of these models, similar to the process used to address RQ3. In addition, the Wilcoxon signed-rank test was used to compare Sensitivity, Specificity, and AUC of the three pairs of metric sets: (I) ENs vs. CMs; (II) GNs vs. CMs; and (III) NMs vs. CMs.

Figure 2 shows the distributions of the Sensitivity, Specificity, and AUC that were obtained from the cross-project predictions. Each box in the boxplot corresponds to the performance of an HSF model built with one of the five metric sets: CMs, ENs, GNs, NMs, and CNs.

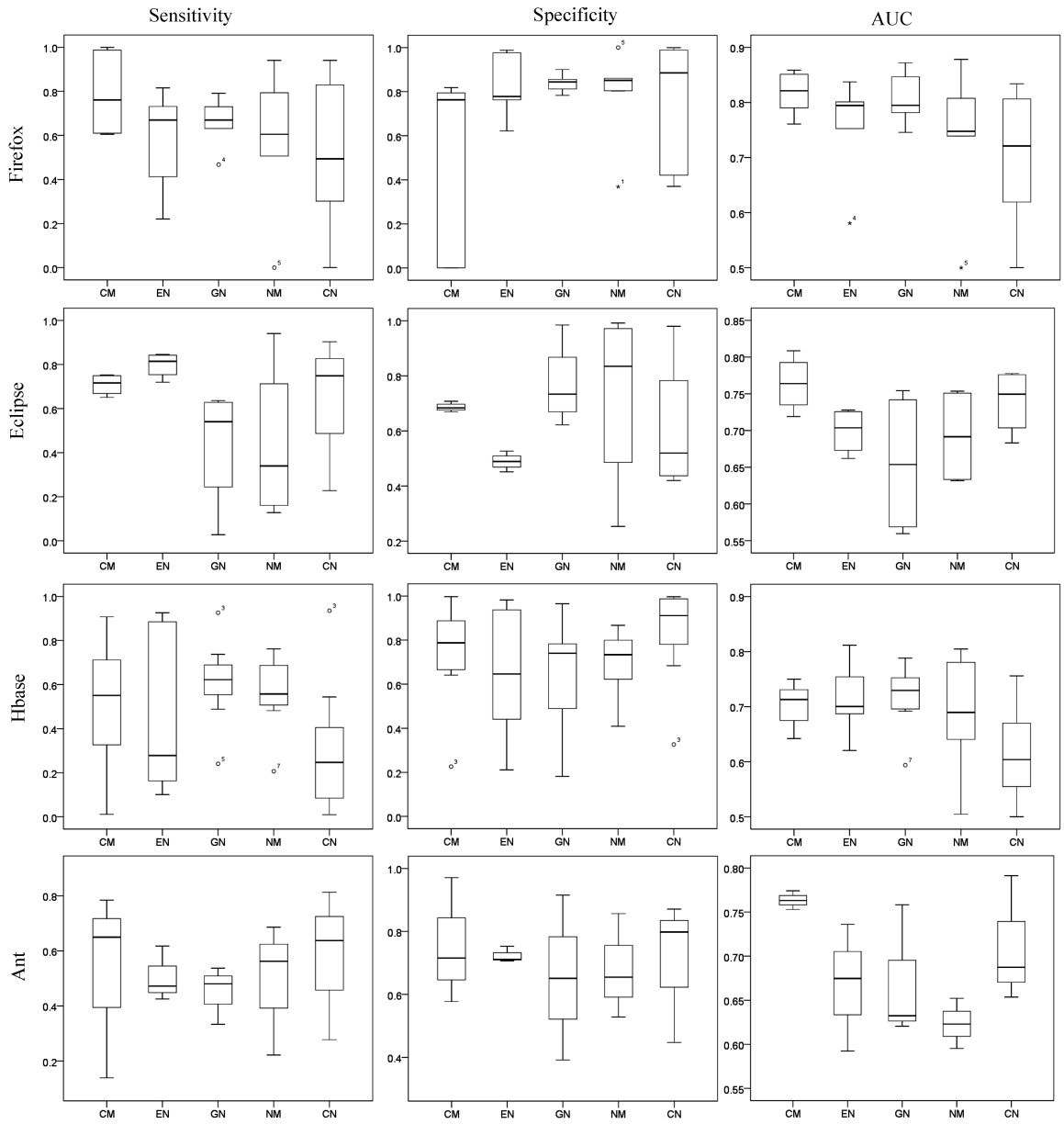

Figure 1 The performance of forward-release predictions for fault-proneness at high severity level.

The HSF models for Firefox 4.0 were transferred to a total number of 17 versions of the other projects to predict the fault-prone modules at a high severity level. The results show that in the cases of the NMs and CNs, all of the modules were classified as fault-prone in most predictions, while the GNs could identify none of the actual faulty modules. This poor performance of the three models is possibly due to the thresholds being inappropriate for use in new projects to determine whether a module was fault-prone at a high severity. The CMs obviously have significantly higher AUC values than the GNs and NMs have. When comparing the ENs with the CMs, the ENs have lower median/mean Sensitivity and AUC values but higher Specificity values. However, the results of the Wilcoxon tests indicate that only the difference in Specificity between the three compared pairs is significant.

The HSF models of Eclipse 2.0 were then used to predict the fault-prone modules at a high severity in 18 versions of the other projects. The three sets of network measures (ENs, GNs and NMs) all show higher mean/median Sensitivity values but lower Specificity values than the CMs show. These metric sets have comparable mean/median AUC values. For Sensitivity and Specificity, the p-values of the Wilcoxon

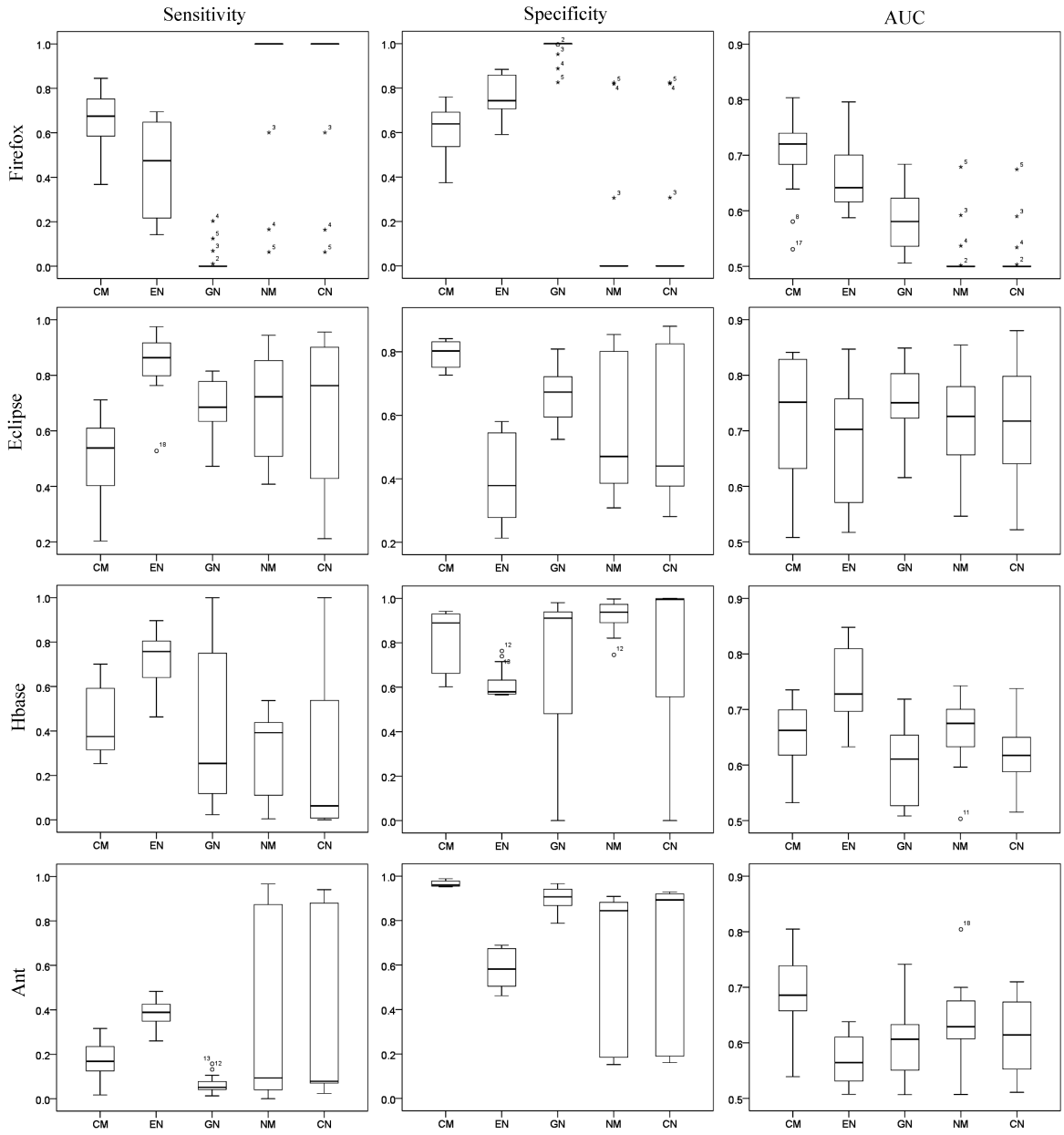

Figure 2 The performance of cross-project predictions for fault-proneness at high severity level.

tests are significant all three compared pairs of metric sets.

Fifteen versions of the other three projects were tested using the HSF models of Hbase 0.20. Compared with the CMs, the ENs have higher mean/median Sensitivity and AUC values but lower Specificity values, while the GNs have lower mean/median Sensitivity and AUC values but higher Specificity vallues. However, the Wilcoxon tests show that only the differences between the ENs and CMs are significant for the three indicators.

The cross-project prediction performance of the HSF models of Ant 1.4 is evaluated on 19 versions of the other projects. Compared to the corresponding values of the CMs, the mean/median Specificity and AUC values of the three sets of network measures are all lower. The ENs show higher mean/median Sensitivity values than the CMs show, while the mean/median Sensitivity values of the GNs and NMs are lower. The differences between the ENs/GNs and CMs are all significant because all of the indicators demonstrate p-values less than 0.05.

Response to hypothesis $H4_0$: "Network measures do not have better portability than code metrics

for fault-proneness at a high severity level." Partially reject. Overall, by combining the results from the four studied projects, we found that most p-values obtained from the Wilcoxon tests were significant. However, the network measures were observed to outperform the code metrics only in some cases.

5 Discussion

5.1 Transitive dependencies are not significantly important

We distinguish two categories of network measures, namely ENs and GNs. The ENs only consider direct dependencies between modules, while the GNs also measure indirect, that is, transitive dependencies. As discussed in Subsections 4.3 and 4.4, the ENs outperform the GNs with significantly higher Sensitivity values in some cases; however, they are not inferior to the GNs in terms of the Specificity or AUC values, and vice versa. No substantial differences in predictive capability of the ENs and GNs were found. Thus, we conclude that direct and indirect dependencies should be treated as equally in fault-proneness prediction. On the other hand, using the ENs and GNs together did not improve the predictive effectiveness. Nevertheless, the combination of the ENs and GNs (i.e., NMs) can enhance the explanatory ability compared to that of the ENs or GNs alone.

5.2 Cross-project predictions are unstable

When performing cross-project predictions, the performance of a classification model may vary significantly when it is applied to different projects. Some models cannot identify any of the fault-prone modules. There are two possible reasons for this inability. First, network measures are more specific to the source code structure than general code metrics. Different software systems have various structures and features. A subset of metrics selected to build the regression model and characterize single project may fail to capture the salient features of other projects. For example, the prediction models built with Firefox performed extremely poorly when transplanted to Ant. This poor performance may have been caused by differences in the source code structure, programming language, and software size. This result provides supplement evidence for the importance of feature selection in defect prediction [31], especially cross-project prediction [32]. However, this conjecture requires more evidence, and we plan to conduct further studies to explore the hidden reasons. Second, software systems vary in quality. The threshold used to determine a healthy project may be too high to identify fault-prone modules in a poor project. Therefore, it may be more suitable to use these models to rank modules according to their predicted probabilities of fault-proneness. Software practitioners could simply select as many potential faulty modules as available resources will allow. Thus, the ranking capability of network measures at a high severity level will be the focus of future research.

5.3 Threats to validity

In this section, we discuss the most important threats to the construct, internal, and external validity of our study following common guidelines for empirical studies [33].

The first threat concerns the manner in which we collected faults and linked them with versioning files. We only selected faults with references in change logs, such as "#623441", which may have led to false negatives in the fault data set because some developers do not leave references for faults in change logs. In addition, we classified each fault by the release corresponding to the date on which the fault was reported. We cannot determine whether this fault was introduced in previous versions, which may result in false negatives. Furthermore, since we are not authorized to access some of the latest security bugs, and thus cannot identify their root causes, we excluded such security bugs from our study. This exclusion may also have increased the number of false negatives. The fault severities in bug reports are not necessarily assessed properly. Users who reported faults may have little professional knowledge; thus, they evaluate the fault severity based on their understanding and experience, which may not comply with

the guidelines. However, Kim et al. [34] discovered that false negatives in fault data do not affect fault prediction performance in a significant manner.

The second threat is that the degree of network construction accuracy depends on the tool used. Although Understand is a mature commercial tool that has been used to collect metric data in many previous studies [35–38], some features of C/C++ cannot be handled perfectly by Understand, such as function calls via pointers. Understand associates dependence between a function and its address. However, table or array-driven setups may take the address of a location other than that where the function is actually used. Nonetheless, Understand can still work fairly well with other types of code. In future, other static tools will be adopted to compare and validate the construction of source code networks.

The third threat to the validity of our study is the unknown effect of deviation of the independent variables from a normal distribution. In logistic regression, there is no assumption related to normal distribution. Therefore, we did not consider whether or not the independent variables followed a normal distribution, and the raw data were used to build the logistic regression models. However, previous studies have suggested that applying a log transformation to the independent variables causes their distributions to approach a normal distribution and may lead to a better model [39]. To eliminate this threat, we applied a log transformation to the network measures and the code metrics of Firefox and reran the analyses. We found that the conclusions for RQ3 and RQ4 did not change significantly before and after the log transformation.

The fourth threat concerns the possibility of generalizing our findings to other systems. We studied multiple versions of four long-lived and popular projects that vary in size, programming language, and application domain. The data sets collected from these systems were large enough to draw statistically meaningful conclusions. We believe that the results of our study significant contribute to empirical software engineering knowledge regarding the usefulness of network measures in fault-proneness predictions that consider severity. Nevertheless, we cannot assume that our results can be generalized beyond the specific environments examined in this study. Therefore, further validation using a larger set of software systems is required.

6 Related work

Zimmermann and Nagappan [13, 14] proposed that network measures on the dependency relationships between binaries of Window Server 2003 were able to predict bug occurrence and numbers. Network measures could identify 60% critical binaries twice as complexity metrics. Tosun et al. [16] reproduced their work on three small scale software systems and Eclipse at function level and source file level. Their results revealed that network measures were effective indicators of defective modules for large and complex systems, but did not provide significant predictive power on small scale projects. In addition, Premraj et al. [17] found that network measures provided no advantage over code metrics for forward-release and cross-project prediction. Nguyen et al. [18] explored the rational behind the better performance using network measures. They also demonstrated that predicting bugs at class level was more useful than at higher level concerning the effort involved. These studies are all about predicting whether there is a bug or not, while our work focuses on the criticality of bugs.

Zhou and Leung [1] made an empirical study on the relationship between object-oriented design metrics and fault proneness of class with fault severity taken into account. They used logistic regression and machine learning methods to investigate the ability of a subset of the Chidamber and Kemerer metrics suite in predicting high and low severity faults of a NASA project. They found these metrics had better prediction capability of low severity faults than high severity ones. Singh et al. [40] conducted a study similar to Zhou and Leung on the same dataset. They studied the effect of object-oriented metrics on fault proneness at three severity levels: high, middle and low. They concluded that the model predicted with respect to middle severity faults had the best performance. Shatnawi and Li [41] also used object-oriented metrics to identify fault-prone classes in Eclipse at high, middle and low severity levels. Chhillar

and Nisha [2] predicted the number of faults at three severity categories based on a NASA dataset. Our study is different from their work where we are interested in the capability of network measures, rather than object-oriented metrics, in predicting fault proneness at high severity level with logistic regression.

7 Conclusion and future work

In this paper, we discussed effectiveness of network measures in fault-proneness prediction at high severity level. By examining four open-source projects, we concluded the paper.

- Most of the network measures are significantly correlated with high severity fault-proneness.
- Generally, when used together, network measures can predict high severity faults effectively as the selected code metrics can, and their predictive powers are comparable to those of the selected code metrics for forward-release high severity fault-proneness prediction.
- Network measures are very unstable for cross-project predictions of high severity faults.

These results indicate, that network measures are of practical use in high severity fault-proneness predictions. Furthermore, the results of our study provide valuable information in an important area with limited experimental data. Our findings could help improve understanding of network measures and guide the development of improved fault-proneness prediction models in practice.

However, in this study we only investigated the actual usefulness of network measures in the context of classifying modules as fault-prone or not fault-prone. Thus, further examination of their ability to rank modules according to the probability of fault-proneness is desired. In addition, the unstable cross-project predictions inspire further research to investigate the underlying causes and to explore how to use network measures to improve the prediction performance.

Acknowledgements This work was partially supported by National Natural Science Foundation of China (Grant Nos. 61472175, 61472178, 61272082, 61272080, 91418202), Natural Science Foundation of Jiangsu Province (Grant No. BK20130014), and Natural Science Foundation of Colleges in Jiangsu Province (Grant No. 13KJB520018). All support is gratefully acknowledged.

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Zhou Y, Leung H. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. IEEE Trans Softw Eng, 2006, 32: 771–789
2 Chhillar R S, Nisha. Empirical analysis of object-oriented design metrics for predicting high, medium and low severity faults using mallows Cp. ACM SIGSOFT Softw Eng Notes, 2011, 36: 1–9
3 Basili V R, Briand L C, Melo W L. A validation of object-oriented design metrics as quality indicators. IEEE Trans Softw Eng, 1996, 22: 751–761
4 Subramanyam R, Krishnan M S. Empirical analysis of ck metrics for object-oriented design complexity: implications for software defects. IEEE Trans Softw Eng, 2003, 29: 297–310
5 Nagappan N, Ball T, Zeller A. Mining metrics to predict component failures. In: Proceedings of the 28th International Conference on Software Engineering. New York: ACM, 2006. 452–461
6 Zhang H Y. An investigation of the relationships between lines of code and defects. In: Proceedings of 2009 IEEE International Conference on Software Maintenance. Piscataway: IEEE, 2009. 274–283
7 Nagappan N, Ball T. Use of relative code churn measures to predict system defect density. In: Proceedings of the 27th International Conference on Software Engineering. Piscataway: IEEE, 2005. 284–292
8 Moser R, Pedrycz W, Succi G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proceedings of the 30th International Conference on Software Engineering. Piscataway: IEEE, 2008. 181–190
9 Hassan A E. Predicting faults using the complexity of code changes. In: Proceedings of the 31st International Conference on Software Engineering. Piscataway: IEEE, 2009. 78–88
10 Hassan A E, Holt R C. The top ten list: dynamic fault prediction. In: Proceedings of the 21st IEEE International Conference on Software Maintenance. Piscataway: IEEE, 2005. 263–272
11 Ostrand T J, Weyuker E J, Bell R M. Predicting the location and number of faults in large software systems. IEEE Trans Softw Eng, 2005, 31: 340–355
12 Zhang W Q, Nie L M, Jiang H, et al. Developer social networks in software engineering: construction, analysis, and applications. Sci China Inf Sci, 2014, 57: 121101

13 Zimmermann T, Nagappan N. Predicting defects with program dependencies. In: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement. Piscataway: IEEE, 2009: 435–438
14 Zimmermann T, Nagappan N. Predicting defects using network analysis on dependency graphs. In: Proceedings of the 30th International Conference on Software Engineering. New York: ACM, 2008. 531–540
15 Taba S E S, Khomh F, Zou Y, et al. Predicting bugs using antipatterns. In: Proceedings of the 29th IEEE International Conference on Software Maintenance. Piscataway: IEEE, 2013. 270–279
16 Tosun A, Turhan B, Bener A. Validation of network measures as indicators of defective modules in software systems. In: Proceedings of the 5th International Conference on Predictor Models in Software Engineering. New York: ACM, 2009. 1–5
17 Premraj R, Herzig K. Network versus code metrics to predict defects: a replication study. In: Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement. Piscataway: IEEE, 2011. 215–224
18 Nguyen T H D, Adams B, Hassan A E. Studying the impact of dependency network measures on software quality. In: Proceedings of the 2010 IEEE International Conference on Software Maintenance. Piscataway: IEEE, 2010. 1–10
19 Ma Y, He K, Li B, et al. How multiple-dependency structure of classes affects their functions a statistical perspective. In: Proceedings of the 2nd International Conference on Software Technology and Engineering. Piscataway: IEEE, 2010, v2: 60–66
20 Basili V R, Shull F, Lanubile F. Building knowledge through families of experiments. IEEE Trans Softw Eng, 1999, 25: 456–473
21 Halstead M H. Elements of Software Science. New York: Elsevier, 1977. 50–70
22 Chidamber S R, Kemerer C F. A metrics suite for object oriented design. IEEE Trans Softw Eng, 1994, 20: 476–493
23 Hosmer Jr D W, Lemeshow S. Applied Logistic Regression. New Jersey: John Wiley & Sons, 2004. 153–223
24 Belsley D A, Kuh E, Welsch R E. Regression Diagnostics: Identifying Influential Data and Sources of Collinearity. New Jersey: John Wiley & Sons, 2005. 6–38
25 Harrell F E. Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis. New York: Springer, 2001. 215–268
26 Kutner M H, Nachtsheim C, Neter J. Applied Linear Regression Models. 4th ed. Chicago: Irwin, 2004. 20–70
27 Maddala G S. Limited-Dependent and Qualitative Variables in Econometrics. New York: Cambridge University Press, 1983. 100–124
28 Nagelkerke N J D. A note on a general definition of the coefficient of determination. Biometrika, 1991, 78: 691–692
29 Benjamini Y, Hochberg Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. J Royal Stat Soc Ser B (Methodological), 1995, 57: 289–300
30 Freeman E A, Moisen G G. A comparison of the performance of threshold criteria for binary classification in terms of predicted prevalence and kappa. Ecol Model, 2008, 217: 48–58
31 He Z, Shu F, Yang Y, et al. An investigation on the feasibility of cross-project defect prediction. Autom Softw Eng, 2012, 19: 167–199
32 Chang R H, Mu X D, Zhang L. Software defect prediction using non-negative matrix factorization. J Softw, 2011, 6: 2114–2120
33 Yin R K. Case Study Research: Design and Methods. 3rd ed. New York: SAGE Publications, 2002. 120–180
34 Kim S, Zhang H, Wu R, et al. Dealing with noise in defect prediction. In: Proceedings of the 33rd International Conference on Software Engineering. Piscataway: IEEE, 2011. 481–490
35 Zhou Y, Xu B, Leung H. On the ability of complexity metrics to predict fault-prone classes in object-oriented systems. J Syst Softw, 2010, 83: 660–674
36 Zhou Y, Leung H, Xu B. Examining the potentially confounding effect of class size on the associations between object-oriented Metrics and change-proneness. IEEE Trans Softw Eng, 2009, 35: 607–623
37 Pan K, Kim S, Whitehead E J. Bug classification using program slicing metrics. In: Proceedings of the 6th International Working Conference on Source Code Analysis and Manipulation, Philadelphia, 2006. 31–42
38 Koru A G, Tian J. Comparing high-change modules and modules with the highest measurement values in two large-scale open-source products. IEEE Trans Softw Eng, 2005, 31: 625–642
39 Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors. IEEE Trans Softw Eng, 2007, 33: 2–13
40 Singh Y, Kaur A, Malhotra R. Empirical validation of object-oriented metrics for predicting fault proneness models. Softw Qual J, 2010, 18: 3–35
41 Shatnawi R, Li W. The effectiveness of software metrics in identifying fault-prone classes in post-release software evolution process. J Syst Softw, 2008, 81: 1868–1882