Abductive Network Ensembles for Improved Prediction of Future Change-Prone Classes in Object-Oriented Software

Mojeeb Al-Khiaty[1], Radwan Abdel-Aal[2], and Mahmoud Elish[1,3]
[1]Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Saudi Arabia
[2]Computer Engineering Department, King Fahd University of Petroleum and Minerals, Saudi Arabia
[3]Computer Science Department, Gulf University for Science and Technology, Kuwait

Abstract: *Software systems are subject to a series of changes due to a variety of maintenance goals. Some parts of the software system are more prone to changes than others. These change-prone parts need to be identified so that maintenance resources can be allocated effectively. This paper proposes the use of Group Method of Data Handling (GMDH)-based abductive networks for modeling and predicting change proneness of classes in object-oriented software using both software structural properties (quantified by the C&K metrics) and software change history (quantified by a set of evolution-based metrics) as predictors. The empirical results derived from an experiment conducted on a case study of an open-source system show that the proposed approach improves the prediction accuracy as compared to statistical-based prediction models.*

Keywords: *Change-proneness, software metrics, abductive networks, ensemble classifiers.*

Received June 2, 2015; accepted September 20, 2015

1. Introduction

It has been a major goal in science to quantify observations in order to understand and harness the underlying phenomena. In this regard, software engineering involves the scientific use of quantitative and qualitative data to understand and improve software and thus produce software with predictable cost and schedule [27]. In a world of constantly changing requirements, software systems are subject to changes. These changes can be due to a variety of maintenance goals, such as adding new features to the system, adapting the system to new environments, fixing bugs, and/or improving the quality of the source code. On the other hand, these changes are perceived as important risk elements as they require time and effort [21,30]. Moreover, maintenance costs account for 90% of the total costs of software systems [11]. Focusing on all software parts equally is difficult [21] and wasteful of resources, especially when systems get larger [21] and more complex [34]. Some parts are more prone to change than others and, as implied by the 80-20 law, the great majority of changes are usually rooted in a small portion of the software system. Resources and effort should be assigned accordingly.

Software-change prediction is one of the fundamental activities with regards to supporting software changes [25]. The process and the methodology of supporting software changes are a decisive factor between the sustained high-quality evolution and the premature retirement of a software system [25]. Therefore, it is pressing to devise methodologies to effectively identify change-prone classes in object-oriented software. Doing so plays a critical role in: reducing the maintenance cost and time, targeting the resources more effectively and efficiently to the most critical parts of the system, and focusing the attention of the developers to those parts that are more prone to changes. As object-oriented metrics provide important evidence about different decision-making activities [7], they can help the software engineer identify the change-prone classes in object-oriented software. Software metrics can be classified into product and evolution-based (process) metrics [16, 20]. Product metrics are those that describe characteristics of the development life cycle processes outputs. They are measures of the software at any stage of its development, from requirements to installed system. Examples of such metrics are size, coupling, and cohesion metrics. Evolution-based metrics, on the other hand, are those that can be computed using data taken from the change history of the software. Age of the class, frequency of changes, are examples of such metrics.

Several approaches have been developed for predicting software changes using software metrics as predictors. Li and Henry [31] have performed statistical analysis of a prediction model incorporating ten object-oriented metrics. Their results showed a strong relationship between these metrics and the maintenance effort measured as the number of lines changed in an object-oriented class. Thwin and Quah [37] described the application of two neural network models (ward neural network and general regression neural network) to predict the number of lines changed per object-oriented class as a proxy measure for the maintenance effort. They have evaluated and compared the application of these two neural network models in predicting software maintenance. Their results showed that the selected object-oriented metrics are good indicators of maintenance effort. Additionally, the result showed that the two neural network models gave comparable results.

Aggarwal *et al*. [4] have presented an application of multilayer feed forward networks to predict the number of lines changed per object-oriented class as a proxy measure for the maintenance effort. The number of lines changed per class was used as a dependent variable while the independent variables were 8 product metrics represented by 3 principal components. They concluded that the selected object-oriented metrics are useful indicators of maintenance effort. Kaur *et al*. [26] have investigated the use of Adaptive Neuro-Fuzzy Inference System to predict the maintenance effort in terms of the number of line changed per object-oriented class. They evaluated and compared the application of this hybrid soft computing technique with other soft computing techniques including Artificial Neural Networks and Fuzzy Inference Systems to construct models for predicting the software maintenance effort. As predictors, they used 8 product metrics represented by 6 principal components. They concluded that Adaptive Neuro-Fuzzy Inference System technique gives the most accurate prediction.

Güneş-Koru and Liu [21] proposed a method for identifying and characterizing change-prone classes using tree-based models. Romano and Pinzger [36] investigated the potential of using a set of source code metrics for predicting change-proneness of Java interfaces using three different classifiers: Support Vector Machine (SVM), Naïve Bayes Network, and Artificial Neural Networks. The dependent binary variable indicated change-proneness depending on whether or not the number of lines changed exceeded the median of the lines changed in all classes of the system. The independent variables are 4 (out of 6) CK metrics [12] as well as the Interface Usage Cohesion (IUC) metric. Their cross validation results showed that the inclusion of the IUC metric improved the classification accuracy of the three classifiers compared to models built using only the 4 CK metrics. However, this improvement was significant only when using SVM as a classifier. The same subset of CK metrics, as well as the other four product metrics, were used in Eski and Buzluca [18] to predict the change-prone classes with the objective of identifying the critical classes for effective testing. Their approach was to rank the classes based on the values of different combinations of metrics and then calculate the correlation between the top 10% of different ranking lists representing the different combinations, and the top 10% of another ranked list representing a set of classes ranked based on the values of dependent variable (measured as the amount of changes between two different version of the class). The metrics combinations corresponding to the ranking list that showed significant correlation were selected as class change-proneness indicators. The best accuracy reported (in terms of correctly identified change-prone classes) over three different case studies was 80%.

To sum up, the literature suggests that product metrics are generally associated with change-proneness. However, the prediction accuracy is still limited. This is due to the fact that relationships between software metrics and quality factors, such as maintainability or changeability, are often complex and sometime nonlinear. Improving the prediction accuracy of change-prone classes is very important, as it should lead to better decision making on resource allocation; save deployment time; and lower development and maintenance costs; thus promoting better evolution management.

In a work by Elish and Al-Khiaty [16], the prediction accuracy of change prone classes has been improved by adding the evolutionary information of the system to its structural properties through a statistical regression modeling. The basic idea was to have a comprehensive view about the system using both its evolution history (quantified by a suite of evolutionary metrics) and its structural properties (quantified by C&K metrics). Towards the same objective, and using the same suites of metrics, this paper proposes using abductive networks [32] based on the Group Method of Data Handling (GMDH) [24] as an alternative approach to model and predict change proneness of classes in object-oriented software. The approach was used previously as a powerful tool in several areas including modeling and forecasting energy consumption and environmental monitoring [1, 2, 3], spam detection [14], intrusion detection [6], and pattern recognition [15,29]. Inspired by promising results obtained in these fields, we explore the use of this approach for the prediction of change-prone classes in object-oriented software. Compared to neural networks, abductive networks offer faster model development requiring little user intervention, faster convergence during model synthesis, avoiding the problem of getting stuck in local minima, and automatic configuration of model structure and selection of effective input variables [23]. Analytical relationships obtained from the resulting polynomial models can provide insight into the modeled phenomena, highlight contributions of various model inputs, and allow comparison with previously used empirical or statistical models.

2. GMDH-based Abductive Networks

The Abductory Inductive Mechanism (AIM) tool [5] is a modern implementation of the GMDH algorithm [24]. The self-organizing modeling tool synthesizes input-output models to represent the structure of complex and nonlinear relationships, automatically selecting the most relevant inputs. The GMDH algorithm uses polynomial regression iteratively to arrive at a high-degree polynomial model in terms of effective predictors. The process is 'evolutionary', starting with simple regression relationships to derive more accurate representations at later iterations. To limit the complexity of the resulting models, only regression relationships with good prediction performance are kept at each phase. In the classical GMDH implementation, such performance is evaluated on a dedicated testing subset of the data. Iteration is stopped when the new generation of regression equations starts to give inferior performance compared to that of the previous generation. At this point, the model starts to over-fit the training data and therefore may not generalize well with new evaluation data. A detailed mathematical treatment of the classical GMDH algorithm can be found in [19].

Several implementations of the GMDH approach have later been proposed which operate on the full training dataset, thus avoiding the need for a dedicated testing subset. One such method is the Adaptive Learning Network (ALN), implemented by AIM. The method uses the Predicted Squared Error (PSE) criterion [19] for selecting promising regression relationships and for stopping the training to avoid over-fitting. This criterion minimizes the squared error expected when using the network to predict new data. AIM expresses the *PSE* as [19]:

$$PSE = FSE + CPM(2K/N)\sigma_p^{\,2} \tag{1}$$

where FSE is the fitting squared error on the training data, *CPM* is a complexity penalty multiplier set by the user, *K* is the number of model coefficients, *N* is the number of training samples, and $\sigma_p^{\,2}$ is a prior estimate of the error variance. With increased model complexity relative to the training set size, the first term in Equation (1) decreases while the second term increases linearly. *PSE* exhibits a minimum at the optimum model size that balances accuracy with simplicity (exactness with generality). By selecting the *CPM* parameter, the user can control this trade-off: *CPM* values above the default value of 1 give simpler models which are less accurate but may generalize well with new evaluation data, while lower values give more complex models that could over-fit the training data and performs poorly on evaluation data previously unseen during training.

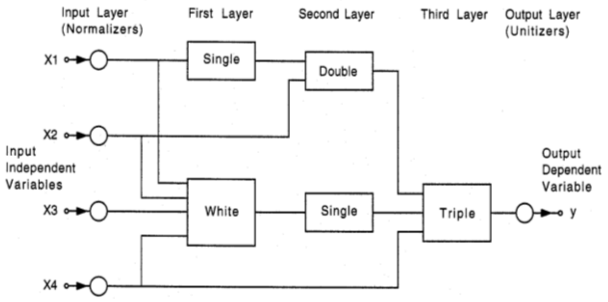


Figure 1. A typical AIM abductive network model showing various types of functional elements.

The AIM tool synthesizes networks of several types of polynomial functional elements. The network size, element types, connectivity, and coefficients for the optimum model are all determined automatically, which reduces required user intervention compared to neural networks. This simplifies model development and considerably reduces the learning/development time and effort. The models take the form of layered feed-forward abductive networks of functional elements (nodes) [5], as shown in Figure 1 [14]. Elements in the first layer operate on various combinations of the input variables (*x's*) and the element in the final layer generates the estimated output. In addition to the main layers of the network, an input layer of normalizers transform the input variables into an internal representation as *Z* scores with zero mean and unity variance. Similarly, a unitizer restores the output to the original problem space. The tool supports the following functional elements:

1. A white element consisting of a constant plus the linear weighted sum of all outputs of the previous layer, i.e., "White".

$$\text{Output}=w_0 + w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n \tag{2}$$

where $x_1, x_2, ..., x_n$ are the inputs to the element and $w_0, w_1, ..., w_n$ are the element weights.

2. Single, doublet, and triplet elements which implement a third-degree polynomial with all possible cross-terms for one, two, and three inputs respectively; e.g., "Doublet".

$$\text{Output}=w_0+w_1x_1+w_2x_2+w_3x_1^2+w_4x_2^2+w_5x_1x_2+w_6x_1^3+w_7x_2^3 \tag{3}$$

3. The dataset

The dataset consists of two suites of metrics, structural-based [12] and evolution-based metrics [16], extracted from a Java-base object-oriented open source software system, VSSPLUGIN [8]. The first suite of metrics (referred to here as C&K) is a well-defined suite of object-oriented metrics in literature. C&K metrics have been theoretically validated [13]. They have also been empirically investigated and found to be associated with various quality aspects [16,18,22].

Additionally, they measure and quantify several structural properties of the system such as size, coupling, cohesion, and inheritance. They have been collected from the open source system using the Understand tool [9], the analyst edition. The second suite of metrics (referred to here as evolutionary) measures and quantifies the change history of the software system. They have been theoretically and empirically validated [16]. They introduce another complementary dimension to understand the evolution and changes in a software system [16]. Both structural-based and evolution-based metrics were found to be associated with several software quality aspects, such as maintainability [30], reliability [28, 33], fault-proneness [36], and change-proneness [16]. Table 1 provides a brief description of each metric. The open source system from which the dataset was collected is a long-lived system, of reasonable size, relatively mature, and of multiple releases (13 releases). Working on a long-lived system prevents results from being biased by the potential data fluctuations experienced during a short period of time [21]. Investigating a reasonable-size system provides a large number of data points, a desirable feature when doing statistical analysis [10], and allows better training and improved performance evaluation [17]. Table 2 shows more quantitative information about the VSSPLUGIN software.

Table 1. Metrics description.

Metric	Name	Definition	Scale
BOC	Birth of the class	The ordinal release number of the release at which the class was introduced.	Ordinal
TACH	Change size	The sum of added lines, deleted lines, and twice changed lines between release n-1 and release n.	Ratio
FCH	First time changes	The ordinal release number at which changes have been introduced to the class for the first time.	Ordinal
LCH	Last time change	The ordinal release number at which most recent changes have been applied to the class.	Ordinal
CHO	Change occurred	A binary metric that indicates whether or not the class has been exposed to changes from release n-1 to n	Nominal (Binary)
FRCH	Frequency of change	The number of times (in term of releases) the class has been changed.	Ratio
WCH	Weighted changes	The aggregated weighted amount of changes in lines of code (added + deleted + twice changed) between each two consecutive releases *r*-1 and *r*, from its first release *j* through release *n*, giving more weight for the latest changes by applying weighting function 2^{r-n} that decreases the importance of a change at the release *r*, which is more distant from the current release *n*.	Ratio
CHD	Change density	The change size (TACH) of the class normalized by the size of the class (its total Lines Of Code (LOC)).	Ratio
WCD	Weighted change density	This metric is similar to the WCH metric, but here, weight is applied to the CHD, instead of the change size (TACH).	Ratio
WFR	Weighted frequency of changes	Aggregated weighted occurrence of changes, favoring the latest occurrence of changes over the old ones.	Ratio
ATAF	Aggregated change size normalized	Aggregated change size of the class over the past releases normalized by frequency of change.	Ratio
LCA	Last change amount	The last change size of the class when moving from release i-1 to release i.	Ratio
LCD	Last change density	The last change size of the class (LCA) normalized by the size of the class.	Ratio
CSB	Changes since the birth	The change size between first version of the class and its current version.	Ratio
CSBS	Changes since the birth normalized	The CSB normalized by the size of the first version of the class.	Ratio
ACDF	Aggregated change density normalized	The aggregated change density of the class over the past releases normalized by frequency of changes.	Ratio
WMC	Weighted methods per class	The static complexity of an individual class. With the assumption that all methods of a class are equally complex, then WMC is the number of local methods.	Ratio
DIT	Depth of inheritance tree of a class	The position of the class in the inheritance hierarchy.	
NOC	Number of children	The number of classes that inherit directly from a class.	
CBO	Coupling between objects	The number of other classes that are coupled to a class either as a client or as a supplier.	Ratio
RFC	Response for a class	The cardinality of the response set of the class. The response set of the class is a set of methods that can potentially be executed in response to a message received by an object of that class.	Ratio
LCOM	Lack of cohesion in methods	A measure of not connected method pairs in a class.	

Because the evolution-based metrics are extracted from the change history of the system, no evolution-based metrics are generated for the first release. Thus, we started the analysis from the second release. In other words, the first release to predict the change-proneness of its classes is release R2. For each release *r*, the evolution-based metrics of each class C are calculated from the change history till release *r*, whereas the C&K metrics are extracted from release *r*. ExamDiff Pro tool was used to compare classes from one release with the next. Comment and blank lines were excluded in class comparison. In this paper, we focus on top-level classes. Only one top-level class is defined in each Java source file. Inner classes were treated as contents of the enclosing top-level classes.

Table 3 lists the data type (integer, real, binary) and the primary statistics (minimum, maximum, and average) for the 22 input metrics for the two classes of cases in the dataset, namely positive cases (proneness=1) and negative cases (proneness=0). Metrics exhibiting larger disparity between the two classes, relative to natural variance, should make good predictors for discriminating between the two classes.

Table 2. Descriptive statistics for VSSPLUGIN software system.

Actual release number	Ordinal release number	Release date	Number of classes in the release	Percentage of changed classes from the previous release to this release (%)
0.8	R1	15-07-2002	36	
0.9	R2	19-07-2002	47	67
0.9.1	R3	30-07-2002	47	4
0.9.2	R4	08-08-2002	56	57
1.0	R5	22-09-2002	68	77
1.2	R6	15-01-2003	95	78
1.2.1	R7	18-01-2003	104	20
1.3.0	R8	08-02-2003	118	51
1.4.0	R9	14-03-2003	140	52
1.4.1	R10	17-04-2003	141	7
1.5.0	R11	21-07-2003	152	33
1.6.1	R12	20-06-2005	170	68
1.6.2	R13	09-09-2007	170	15
Max			170	78
Min			36	4
Average			103.38	44

The 1138 cases of the dataset included 696 negative cases and 442 positive cases (amounting to 38.84% of the total dataset population). The dataset was randomized and then split into a training set and an evaluation set using the 70:30 rule, respectively. Therefore, the training set consisted of 797 cases of which 488 cases were negative and 309 cases were positive (38.77% of the total training dataset). The evaluation set consisted of 341 cases of which 208 cases were negative and 133 cases were positive (39.00% of the total evaluation dataset). The random split ensured nearly identical distribution for the Proneness output parameter in the training and evaluation datasets.

Table 3. Data types and primary statistics for the 22 input metrics in the complete dataset for each of the two proneness categories (696 negative cases; proneness=0 and 442 positive cases; proneness=1).

Metric Category	Metric		Data Type	Minimum		Maximum		Average	
	#	Symbol		Proneness =0	Proneness =1	Proneness =0	Proneness =1	Proneness =0	Proneness =1
Evolutionary	1	BOC	Integer	1	1	12	12	5.11	4.22
	2	FCH	Integer	0	0	12	12	2.99	3.17
	3	FRCH	Integer	0	0	10	10	1.15	2.11
	4	LCH	Integer	0	0	12	12	3.96	5.14
	5	WCH	Real	0	0	1059.98	991.00	22.28	52.64
	6	WCD	Real	0	0	12.79	6.81	0.31	0.50
	7	WFR	Integer	0	0	10	10	0.99	1.94
	8	TACH	Integer	0	0	905	990	15.80	30.57
	9	ATAF	Real	0	0	306.50	496.00	18.03	40.57
	10	CHD	Real	0	0	12.71	5.93	0.19	0.29
	11	LCA	Integer	0	0	905	990	21.75	42.55
	12	LCD	Real	0	0	12.71	12.71	0.39	0.47
	13	CSB	Integer	0	0	2919	2425	56.79	130.70
	14	CSBS	Real	0	0	13.89	13.89	0.96	1.44
	15	ACDF	Real	0	0	8.44	8.44	0.36	0.49
	16	CHO	Binary	0	0	1	1	0.28	0.47
C&K	17	LCOM	Integer	0	0	100	100	33.64	50.44
	18	DIT	Integer	0	0	4	4	1.74	1.85
	19	CBO	Integer	0	0	32	31	2.67	4.49
	20	NOC	Integer	0	0	24	24	0.62	0.60
	21	RFC	Integer	0	0	82	81	13.91	17.10
	22	WMC	Integer	0	0	82	81	6.71	10.45

4. Single Abductive Classifiers for Predicting Proneness

We developed three single classification models to predict proneness through training on the training set using three categories of the input metrics. These categories are:

1. The evolutionary set of metrics (first 16 in Table 1).
2. The C&K set of metrics (last 6 in Table 1).
3. All 22 metrics of the two sets combined.

Each of the three models was optimized to minimize the Mean Squared Error (MSE) between the true binary proneness and the continuous predicted proneness value by trying five levels of model complexity corresponding to the following values of AIM's Complexity Penalty Multiplier (CPM): 0.2, 0.5, 1.0, 2.0, and 5.0. The continuous (0-1) Proneness output for each of the optimum models was converted to a binary output through simple rounding at the threshold value of 0.5. Table 4 shows the optimal CPM value, the structure of the synthesized optimal model, a list of the selected input metrics, and the overall classification accuracy when the model was evaluated on the evaluation set. The table shows the significant data reduction achieved by the automatic selection of relevant input metrics during training. A classification accuracy of 72.73% was obtained by both models using the evolutionary metrics and the C&K metrics, which suggests superior quality of the 6 metrics. Best performance (73.02%) was obtained with the model using all 22 metrics, which is consistent with the findings obtained in [16]. This model selects only 5 of the input metrics, namely BOC, FCH, LCOM, LCH, and CHO, thus ignoring 77.3% of the available input metrics. This leads to simpler and more transparent models and highlights the most effective proneness predictors. Only LCOM belongs to the group of the C&K metrics. In light of the Principal Component Analysis (PCA) provided in [16], the 5 selected metrics cover three different dimensions. These dimensions, as categorized in [16], are: class age and change frequency dimensions (covered by BOC); change occurrence dimension (covered by FCH, LCH, and CHO metrics); and size, coupling, and cohesion dimension (covered by LCOM). Unlike [16], the classification models here are not built on a release-by-release basis. Instead, we used all the data collected over the different releases as a combined input to the abductive network to build the classifier. The selected metrics here are compared against the metrics obtained in [16]. Except for CHO, the selected metrics by the abductive network model here are also among the frequently selected metrics by the regression models built on release-by-release basis. As for the 7 metrics selected out of the evolutionary metrics, they cover all the four evolutionary-based dimensions used in [16] based on the PCA analysis. In the model based on the C&K metrics, only one metric (WMC) is not selected. This unselected metric falls in the same dimension covered by LCOM, CBO, and RFC according to the PCA analysis in [16].

5. Ensemble Abductive Classifiers for Predicting Proneness

In an attempt to improve proneness prediction beyond that achieved by the single models described in Section 4, we have investigated combining these models by fusing their outputs to form an ensemble or a committee as shown in Figure 2. Two fusion approaches were investigated for combining the three outputs:

a. Simple averaging of the linear outputs followed by rounding at the threshold of 0.5. This achieved a classification accuracy of 74.49% over the evaluation set, an improvement of 1.47 percentage points over the best single model trained using all 22 metrics.
b. Majority voting of the binary outputs of the three models. This approach proved to be more effective than simple averaging of the linear outputs, leading to 83.58% accuracy. This surpasses the accuracy of the best member of the ensemble by 10.85 percentage points. The improvement highlights the significant advantage of fusing multiple models at the decision level compared to fusion at the metric level as performed by the model employing all 22 metrics.

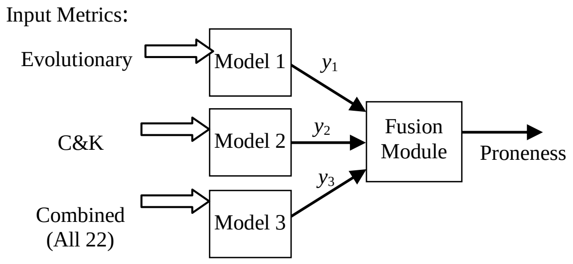
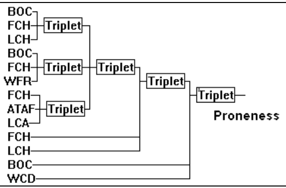
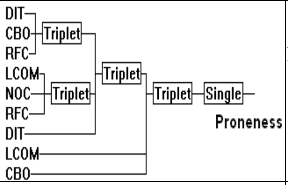
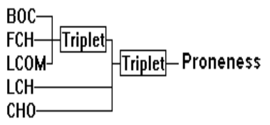


Figure 2. Schematic diagram of the 3-member network ensemble.

Table 4. Optimal structures of the proneness models obtained using the three groups of input metrics.

Metrics Used	CPM	Model	Metrics Selected by the Model	Classification Accuracy (%)
Evolutionary	0.5	 BOC FCH LCH BOC FCH WFR FCH ATAF LCA FCH LCH BOC WCD Triplet Triplet Triplet Triplet Triplet Triplet Proneness	BOC, FCH, LCH, WFR, ATAF, LCA, WCD	72.73
C&K	1.0	 DIT CBO RFC LCOM NOC RFC DIT LCOM CBO Triplet Triplet Triplet Triplet Single Proneness	DIT, CBO, RFC, LCOM, NOC	72.73
Combined (All 22)	5.0		BOC, FCH, LCOM, LCH, CHO	73.02

6. Performance Evaluation

Performance of the proneness classifiers on the evaluation set was measured using the following five metrics:

- Classification accuracy (%), defined as the portion of the total size of the evaluation set (N=341 cases) that was correctly classified. Let TP be the number of true positives (cases in the evaluation set having proneness=1 which were classified as positive, i.e., predicted proneness=1) and TN be the number of true negatives (cases in the evaluation set having proneness=0 which were classified as negative, i.e., predicted proneness=0). The classification accuracy is given by:

$$\text{Accuracy} = 100\frac{(TP+TN)}{N}$$

- Precision (%), defined as the portion of actual positives (proneness=1) in the evaluation dataset (N_p=133 cases) that was correctly classified as predicted proneness=1.

$$\text{Precision} = 100\frac{TP}{N_p}$$

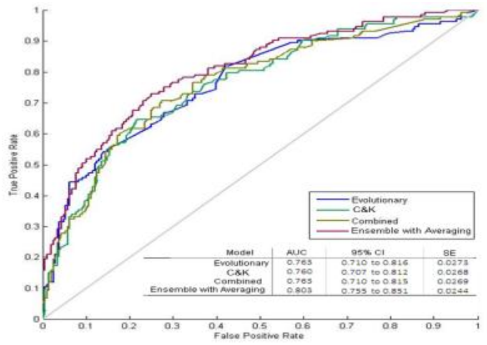
- Recall (%), defined as the portion of all evaluation set cases classified as positive (predicted proneness=1) which are true positives (proneness=1).

$$\text{Recall} = 100\frac{TP}{TP+FN}$$

  where FN is the number of positive cases (proneness=1) classified wrongly as Negative (predicted proneness=0).
- F_1-measure (%), defined as the harmonic mean of precision and recall.

$$F_1 = \frac{2(\text{Precison})(\text{Recall})}{\text{Precison}+\text{Recall}}$$

- The Receiver Operating Characteristics (ROC) is a plot of the true positive rate versus the false positive rate as the rounding threshold used with the continuous classifier output is varied in increments over the interval 0 to 1. The closer this curve gets to the point at which false positive rate is 0 and the true positive rate is 1 the better the classifier performance becomes. Another related parameter is the Area Under the ROC Curve (AUC) (0≤AUC≤1). Larger AUC values indicate better classifier performance.



Model	AUC	95% CI	SE
Evolutionary	0.763	0.710 to 0.816	0.0273
C&K	0.760	0.707 to 0.812	0.0268
Combined	0.763	0.710 to 0.815	0.0269
Ensemble with Averaging	0.803	0.755 to 0.851	0.0244

Figure 3. ROC curves and the AUC values for the three single abductive models and the ensemble model that employs simple averaging of the linear outputs of these single models.

Table 5. Performance metrics for the three optimal single models and the two ensemble models when evaluated on the evaluation set.

Metric	**Optimal Single Models**			**Ensemble Models**	
	Evolutionary	**C&K**	**Combined**	**Averaging**	**Majority Voting**
Precision, %	48.87	62.41	54.89	57.14	68.42
Recall, %	72.22	65.87	69.52	71.70	86.67
F1-Measure, %	58.30	64.09	61.34	63.60	76.47
Accuracy, %	72.73	72.73	73.02	74.49	83.58

Table 5 lists the first four performance metrics for the 3 single and 2 ensemble models described in Sections 4 and 5. The majority voting ensemble model surpasses all other models in all performance aspects, achieving a recall of 86.67%. This means that approximately 86.7% of the predictions of positive proneness made by this model will be correct, which, in turn, means that 86.7% of the maintenance effort will be correctly directed to the change-prone classes - usually a small portion of the overall classes of the system [21, 35]. Comparing the accuracy obtained by the best abductive ensemble classifier here to the accuracy obtained by the regression based classifier in [16], the abductive classifier, as shown in Table 5, achieved a correct classification rate of 83.58%, which is 3.08% higher than the classification accuracy obtained in [16].

Figure 3 plots the ROC curves for the 3 single models and the simple averaging ensemble model described in Section 5. The ROC plots for the 4 models plotted match expectations based on their relative performance, with the curve for the ensemble model being generally the top-most curve. The figure also gives the AUC values for the 4 models, the 95% Confidence Interval (CI), and the Standard Error (SE). The ensemble classifier with averaging has an AUC of 0.803, compare to 1.0 for the ideal classifier. Generating the ROC plot requires the availability of the continuous classifier output to be able to change the threshold at small intervals from 0 to 1 and calculate the corresponding values of the false positive and true positive rates at each threshold value. Since such linear output is not available for the majority voting ensemble model, it is not included in the plot.

7. Conclusions

In this work, we investigated the use of the GMDH-based abductive networks to improve the prediction accuracy of change proneness of classes in object-oriented software. Several prediction models were built using 3 different types of metrics as predictors: (1) evolution-based metrics; (2) C&K metrics; and (3) a combination of these two metrics types. The prediction accuracy has been reported for each single model as well as the fusion of the outputs of the three models to form an ensemble prediction model. The major findings of the conducted empirical investigation are as follows. First, the two set of metrics (evolutionary set, and C&K) are competitive predictors of change proneness. Second, combining the two sets of metrics as inputs to a GMDH-based abductive classifier improves classification accuracy compared to that obtained with a single set of predictors. Third, the best performance (83.58% classification accuracy) was obtained when fusing the outputs of the three single models using majority voting to form an ensemble GMDH abductive classifier. This highlights an accuracy improvement of 10.85% over the accuracy of the best member of the ensemble, and an accuracy improvement of 3.08% as compared to the regression-based classifier presented in [16] using the same dataset.

Future work would explore the potential for using GMDH abductive networks for predicting other software quality aspects such as change size and fault-proneness.

References

[1] Abdel-Aal R., “Hourly temperature forecasting using abductive networks,” *Engineering Applications of Artificial Intelligence*, vol. 17, no. 5, pp 543-556, 2004.

[2] Abdel-Aal R., Al-Garni A., and Al-Nassar Y., “Modelling and Forecasting Monthly Electric Energy Consumption in Eastern Saudi Arabia using Abductive Networks,” *Energy*, vol. 22, no. 9, pp. 911-921, 1997.

[3] Abdel-Aal R., Elhadidy M., and Shaahid S., “Modeling and Forecasting the Mean Hourly Wind Speed Time Series using GMDH-Based Abductive Networks,” *Renewable Energy*, vol. 34, no. 7, pp. 1686-1699, 2009.

[4] Aggarwal K., Singh Y., Kaur A., and Malhotra R., “Application of Artificial Neural Network for Predicting Maintainability using Object-Oriented Metrics,” *Transactions on Engineering, Computing and Technology*, vol. 15, pp. 285-289, 2006.

[5] AIM Software User's Manual, AbTech Corporation, *AIM Software User's Manual*, Charlottesville, 1990.

[6] Baig Z., Sait S., and Shaheen A., “GMDH-Based Networks for Intelligent Intrusion Detection,” *Engineering Applications of Artificial Intelligence*, vol. 26, no. 7, pp. 1731-1740, 2013.

[7] Basili V., Briand L., and Melo W., “A Validation of Object-Oriented Design Metrics as Quality Indicators,” *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751-761, 1996.

[8] Bayer J., Girard J., Wurthner M., DeBaud J., and Apel M., “Transitioning Legacy Assets to a Product Line Architecture,” *in Proceeding of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering*, Toulouse, pp. 446-463, 1999.

[9] Bernstein P., Halevy H., and Pottinger R., “A Vision for Management of Complex Models,” *ACM Sigmod Record*, vol. 29, no. 4, pp. 55-63, 2000.

[10] Boslaugh S. and Watters P., *Statistics in a Nutshell: A Desktop Quick Reference*, O’Reilly Media, 2008.

[11] Brooks F., *The Mythical Man-Month*, Addison Wesley, 1995.

[12] Chidamber S. and Kemerer C., “Towards a Metrics Suite for Object Oriented Design,” *in Proceeding of Object-oriented programming systems, languages, and applications*, phoenix, pp. 197-211, 1991.

[13] Chidamber S. and Kemerer C., “A Metrics Suite for Object Oriented Design,” *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, 1994.

[14] El-Alfy E. and Abdel-Aal R, “Using GMDH-Based Networks for Improved Spam Detection and Email Feature Analysis,” *Applied Soft Computing*, vol. 11, no. 1, pp. 477-488, 2011.

[15] El-Alfy E. and Abdel-Aal R., “Abductive Learning Ensembles for Hand Shape Identification,” *Cognitive Computation*, vol. 6, no. 3, pp. 321-330, 2014.

[16] Elish M. and Al-Khiaty M., “A Suite of Metrics for Quantifying Historical Changes to Predict Future Change-Prone Classes in Object-Oriented Software,” *Journal of Software: Evolution and Process*, vol. 25, no 5, pp. 407-437, 2012.

[17] Ellis D. and Morgan N., “Size Matters: An Empirical Study of Neural Network Training for Large Vocabulary Continuous Speech Recognition,” *in Proceeding of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Phoenix , pp. 1013-1016, 1999.

[18] Eski S. and Buzluca F., “An Empirical Study on Object-Oriented Metrics and Software Evolution in Order to Reduce Testing Costs by Predicting Change-Prone Classes,” *in Proceeding of Software Testing, Verification and Validation Workshops*, honolulu, pp. 566-571, 2011.

[19] Farlow S., *The GMDH algorithm. Self-organizing methods in modeling: GMDH Type Algorithms*, Marcel Dekker, 1984.

[20] Fenton N. and Pfleeger S., *Software Metrics: A Rigorous and Practical Approach*, PWS Publishing Co., 1998.

[21] Güneş-Koru A. and Liu H., “Identifying and Characterizing Change-Prone Classes in Two Large-Scale Open-Source Products,” *Journal of Systems and Software*, vol. 80, no. 1, pp. 63-73, 2007.

[22] Gyimothy T., Ferenc R., and Siket I., “Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction,” *IEEE Transactions on Software Engineering*, vol 31, no. 10, pp. 897-910, 2005.

[23] Hippert H., Pedreira C., and Souza R., “Neural Networks for Short-Term Load Forecasting: A Review and Evaluation,” *IEEE Transactions on Power Systems*, vol. 16, no. 1, pp. 44-55, 2001.

[24] Ivakhnenko A., “Polynomial Theory of Complex Systems,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-1, no. 4, pp. 64-378, 1971.

[25] Kagdi H. and Maletic J., “Software-Change Prediction:Estimated+Actual,” *in Proceeding of Second International IEEE Workshop on Software Evolvability*, Philadelphia, pp. 38-43, 2006.

[26] Kaur A., Kaur K., and Malhotra R., “Soft Computing Approaches for Prediction of Software Maintenance Effort,” *International Journal of Computer Applications*, vol. 1, no. 16, pp. 69-75, 2010.

[27] Kemerer C. and Slaughter S., “An Empirical Approach To Studying Software Evolution,” *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 493-509, 1999.

[28] Khoshgoftaar T., Allen E., Halstead R., Trio G., and Flass R., “Process Measures for Predicting Software Quality,” *in Proceeding of High-*

Assurance Systems Engineering Workshop, Washington, pp. 155-160, 1997.

[29] Lawal I., Abdel-Aal R., and Mahmoud S., "Recognition of Handwritten Arabic (Indian) Numerals Using Freeman's Chain Codes and Abductive Network Classifiers," *in Proceeding of 20th International Conference on Pattern Recognition*, Istanbul, pp. 1884-1887, 2010.

[30] Lehman M., Perry D., and Ramil J., "Implications of Evolution Metrics on Software Maintenance," *in Proceeding of International Conference on Software Maintenance*, Bethesda, pp. 208-217, 1998.

[31] Li W. and Henry S., "Object-Oriented Metrics that Predict Maintainability," *Journal of Systems and Software*, vol. 23, no. 2, pp. 111-122, 1993.

[32] Montgomery G. and Drake K., "Abductive Reasoning Networks," *Neurocomputing*, vol. 2, no. 3, pp. 97-104, 1991.

[33] Nagappan N. and Ball T., "Use of Relative Code Churn Measures to Predict System Defect Density," *in Proceeding of 27th International Conference on Software Engineering*, Louis, pp. 284-292, 2005.

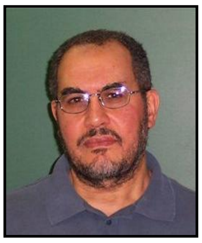
[34] Parnas D., "Software Aging," *in Proceeding of 16th International Conference on Software Engineering*, Sorrento, pp. 279-287, 1994.

[35] Porter A. and Selby R., "Empirically Guided Software Development using Metric-Based Classification Trees," *IEEE Software*, vol. 7, no. 2, pp. 46-54, 1990.

[36] Romano D. and Pinzger M., "Using Source Code Metrics to Predict Change-Prone Java Interfaces," *in Proceeding of 27th IEEE International Conference on Software Maintenance*, Williamsburg, pp. 303-312, 2011.

[37] Thwin M. and Quah T., "Application of Neural Networks for Estimating Software Maintainability Using Object-Oriented Metrics," *in Proceeding of International Conference on Software Engineering and Knowledge Engineering*, San Francisco, pp. 69-73, 2003.

Mojeeb AL-Khiaty received his BS degree in Mathematics and Computer from Sana'a University, Yemen, in 1999, his MS degree in Computer Science from King Fahd University of Petroleum and Minerals (KFUPM), Saudi Arabia, in 2009, and his PhD degree in Computer Science and Engineering from KFUPM, Saudi Arabia, in 2015. His research interests include software engineering, software metrics, software reuse, and soft computing.



Radwan Abdel-Aal received his BS in electrical engineering from Cairo University, Egypt, in 1972, his MS in aviation electronics from Cranfield University, UK in 1974, and his PhD from Strathclyde University, UK in 1983. Between 1985 and 2005 he was a research scientist at the Research Institute of King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia. In 2005 he joined the Computer Engineering Department at KFUPM where he is currently a Professor. His research interests include nuclear physics instrumentation and machine learning and data mining applications.

Mahmoud Elish is an Associate Professor in the Computer Science Department at Gulf University for Science and Technology (GUST), Kuwait. He has been an Associate Professor in the Information and Computer Science Department at King Fahd University of Petroleum and Minerals (KFUPM), Saudi Arabia. He received his PhD from George Mason University. His research interests include software metrics, design, quality and maintenance.