Multi-Objective Cross-Project Defect Prediction

Gerardo Canfora[1], Andrea De Lucia[2], Massimiliano Di Penta[1],
Rocco Oliveto[3], Annibale Panichella[2], Sebastiano Panichella[1]
[1]*University of Sannio, Via Traiano, 82100 Benevento, Italy*
[2]*University of Salerno, Via Ponte don Melillo, 84084 Fisciano (SA), Italy*
[3]*University of Molise, Contrada Fonte Lappone, 86090 Pesche (IS), Italy*
{*canfora, dipenta, spanichella*}*@unisannio.it,* {*adelucia, apanichella*}*@unisa.it, rocco.oliveto@unimol.it*

***Abstract*—Cross-project defect prediction is very appealing because (i) it allows predicting defects in projects for which the availability of data is limited, and (ii) it allows producing generalizable prediction models. However, existing research suggests that cross-project prediction is particularly challenging and, due to heterogeneity of projects, prediction accuracy is not always very good.**

This paper proposes a novel, multi-objective approach for cross-project defect prediction, based on a multi-objective logistic regression model built using a genetic algorithm. Instead of providing the software engineer with a single predictive model, the multi-objective approach allows software engineers to choose predictors achieving a compromise between number of likely defect-prone artifacts (effectiveness) and LOC to be analyzed/tested (which can be considered as a proxy of the cost of code inspection).

Results of an empirical evaluation on 10 datasets from the Promise repository indicate the superiority and the usefulness of the multi-objective approach with respect to single-objective predictors. Also, the proposed approach outperforms an alternative approach for cross-project prediction, based on local prediction upon clusters of similar classes.

***Keywords*-Cross-project defect prediction, multi-objective optimization, search-based software engineering.**

I. Introduction

Defect prediction consists of identifying likely defect-prone software components, in order to prioritize quality assurance activities. Indeed, effective defect prediction models can help developers to focus activities such as code inspection or testing on likely defect-prone components, thus optimizing the usage of resources for quality assurance.

Existing defect prediction models try to identify defect-prone artifacts based on process or product metrics. For example, Basili *et al.* [1] and Gyimothy *et al.* [2] use Chidamber and Kemerer (CK) metrics [3] to build defect prediction models based on logistic regression. Moser *et al.* [4] use process metrics, e.g., related to the number and kinds of changes occurred on software artifacts. Ostrand *et al.* [5] and Kim *et al.* [6] perform prediction based on knowledge about previously occurred faults.

Building an *effective* (able to identify the largest proportion of defect-prone artifacts) and *efficient* (limiting the number of false positives, thus not wasting the developers' effort in the quality assurance task) bug prediction model requires the availability of enough accurate data about a project and its history, as suggested by Nagappan *et al.* [7]. This clearly limits the adoption of prediction models on new projects.

Several authors [8], [9], [10], [11] have discussed the possibility to use data from other projects to train machine learning models to deal with the limited availability of training data. This strategy is referred to as *cross-project defect prediction*. However, works such as those by Turhan *et al.* [8] and by Zimmermann *et al.* [9] showed that cross-project defect prediction does not always work well. The reasons are mainly due to the projects' heterogeneity, in terms of domain, source code characteristics (e.g., classes from a project can be intrinsically larger or more complex than those of another project), and process organization (one project exhibit more frequent changes than another). A noticeable way to tackle the problem of heterogeneity among projects is to perform local prediction [10], [11], i.e., grouping together artifacts belonging to different projects[1].

Recently, Rahman *et al.* [12] pointed out that cross-project defect predictors do not necessarily work worse than within-project predictors. Instead, such models are often quite good in terms of the cost-effectiveness that one can achieve when using them. Specifically, they allow to pursue a good compromise between the number of defect-prone artifacts that the model predict, and the amount of code—i.e., LOC of the artifact predicted as defect-prone—that a developer shall analyze/test to discover such defects.

Stemming from the considerations by Rahman *et al.*—who analyzed the cost-effectiveness of a single objective prediction model—we propose to shift from the single-objective defect prediction model—which recommends a set or a ranked list of likely defect-prone artifacts and tries to achieve a compromise between cost and effectiveness—towards multi-objective defect prediction models. Such models provide the software engineer with a set of predictors, whose performance are Pareto-efficient [13] in terms of cost-effectiveness. In other words, the multi-objective approach allows software engineers to choose between predictors

[1]Although local prediction is particularly useful for cross-project prediction, it can also be used to improve performances of within-project prediction.

able to identify a high number of defect-prone artifacts (however requiring a high cost in terms of amount of code to be analyzed/tested), predictors requiring a lower cost (however recommending a lower number of likely defect-prone artifacts), and predictors achieving a cost-effectiveness compromise.

In order to reach such a goal, we use a multi-objective Genetic Algorithm (GA) to train a logistic regression predictor, which uses product metrics (e.g., size metrics such as LOC, McCabe cyclomatic complexity, and object-oriented metrics such as Chidamber and Kemerer Metrics [3]) as independent variables. Rather than building a model achieving a compromise between inspection cost and number of identified defect-prone artifacts (or cost-effectiveness), the GA evolves the logistic regression coefficients to build Pareto fronts of predictors that (near) optimize the two conflicting objectives of cost and effectiveness. We choose to adopt a cost-effectiveness multi-objective predictor rather than a precision-recall multi-objective predictor because—and in agreement with Rahman *et al.* [12]—we believe that cost is a more meaningful (and of practical use) information to software engineers than precision.

We applied the proposed approach on 10 datasets from the Promise[2] repository. Results of the study show that multi-objective prediction not only allows the software engineer to choose among predictors balancing between different objectives, but it also allows to achieve better results—for cross-project prediction—than single-objective predictors. Finally, we show how the proposed approach outperforms a local prediction approach based on clustering proposed by Menzie *et al.* [10].

The rest of this paper is organized as follows. Section II overviews related work on cross-project defect prediction, while Section III describes the multi-objective defect-prediction approach proposed in this paper. Section IV describes the empirical study we carried out to evaluate the proposed approach. Results are reported and discussed in Section V, while Section VI discusses the study threats to validity. Section VII concludes the paper and outlines directions for future work.

II. Related Work

To the best of our knowledge, the earliest work on cross-project defect prediction is by Briand *et al.* [14]. They build a logistic regression model on the defect data of one open source project and try to use the learned predicting model on the defect data of another open source project, which was developed by an identical team with different design strategies and coding standards. They noted that, even if the development team is the same, cross-project prediction generally exhibits low performances if compared to within-project prediction.

[2]https://code.google.com/p/promisedata/

Turhan *et al.* [8] made a thorough analysis of cross project prediction using 10 projects collected from two different data sources. They propose to use a nearest-neighbor technique for selecting a similar set of training data, filtering out the irrelevancies in cross-project data. Even if the cross-project performances are improved, they are still significantly lower than within-project prediction.

Zimmermann *et al.* [9] considered different factors that may affect the accuracy of cross-project predictions. They categorized projects according to their domain, process characteristics and code measures. In their initial study, they tried to predict defects in Internet Explorer (IE) using models trained with Mozilla Firefox, and *vice versa*. They found that the cross-project prediction was still not consistent: Firefox can predict the defects in IE successfully, while prediction does not work well in the opposite direction.

Cruz *et al.* [15] trained a defect prediction model with an open source project (Mylyn) and test the accuracy of the same model on six other projects. However, before training and testing the model, they try to obtain similar distributions in training and test samples through a data transformation (i.e., power transformation). They observed that using transformed training and test data yields better cross-project prediction performances.

Menzies *et al.* [10] found that prediction accuracy may not even be generalizable within a project. Specifically, data from a project may have many local regions which, when aggregated at a global level, may offer a completely different conclusion in terms of both quality control and effort estimation. For this reason, they proposed a "local prediction" that could be applied to perform cross-project or within-project defect prediction. In the cross-project defect prediction scenario, let us suppose we have two projects, A and B, and suppose one wants to perform defect prediction in B based on data available for A. First, the approach proposed by Menzies *et al.* [10] clusters together similar classes (according to the set of identified predictors) into n clusters. Each cluster contains classes belonging from A and classes belonging from B. Then, for each cluster, classes belonging from A are used to train the prediction model, which is then used to predict defect-proneness of classes belonging from B. This means that n different prediction models are built. The results of an empirical evaluation indicated that conclusions derived from local models are typically superior and more insightful than those derived from global models. The approach proposed by Menzies *et al.* [10] is used in our paper as baseline in our experimentation.

A wider comparison of global and local models has been performed by Bettenburg *et al.* [11], who compared (i) local models, (ii) global models, and (iii) global models accounting for data specificity. Results of their study suggest that local models are valid only for specific subsets of data, whereas global models provide trends too general to be used in the practice.

All these studies suggested that cross-project prediction is particularly challenging and, due to the heterogeneity of projects, prediction accuracy might be poor in terms of precision, recall and F-score. Rahman *et al.* [12] argued that such measures—while broadly applicable—are not well-suited for the quality-control settings in which defect prediction models are used. They show that, instead, the choice of prediction models should be based on both effectiveness (e.g., precision and recall) and inspection cost, which they approximate in terms of number of source code lines that need to be inspected to detect a given number/percentage of defects. By considering such factors, they found that the accuracy of cross-project defect prediction are adequate, and comparable to within-project prediction from a practical point of view.

Our work is inspired by the work by Rahman *et al.* in the use of cost-effectiveness instead of precision and recall. However, while Rahman *et al.* considered cost-effectiveness to assess the quality of a predictor, they still built a single-objective predictor based on logistic regression which, by definition, finds a model that minimizes the fitting error of precision and recall, thus achieving a compromise between them. In this paper we introduce, for the first time, an explicit multi-objective definition of the defect prediction problem, which allows to produce a Pareto front of (near) optimal prediction models—instead of a single model as done in the past—with different recall and cost values. This allows the software engineer to choose the model that better fits her needs and available resources.

III. The Proposed Approach

Our approach builds defect prediction models as follows. First, a set of predictors (e.g., product [1], [2] or process metrics [4]) is computed on each component of a software system. The computed data is then preprocessed to reduce the effect of data heterogeneity. Such preprocessing step is particularly useful when performing cross-project defect prediction, as data from different projects—and in some case in the same project—have different properties [10]. Since a prediction model generally does not explicitly consider the local differences between different software projects when it tries to predict defects across projects, its performances can be unstable [10].

Once the data have been preprocessed, a machine learning technique (logistic regression in our case, however other techniques can be applied without loss of generality) is used to build a prediction model.

A. Data Preprocessing

To reduce the effect of heterogeneity between different projects, in this paper we perform a data *standardization*, i.e., we convert metrics into a z distribution. Given the value of the metric m_i computed on the software component s_j of project S, denoted as $m(i,j,S)$, we convert it into:

$$m_z(i,j,S) = \frac{m(i,j,S) - \mu(i,S)}{\sigma(i,S)}$$

In other words, we subtract from the value of the metric m_i the mean value $\mu(i,S)$ obtained across all components of system P, and divide by the standard deviation $\sigma(i,S)$. A similar approach has been applied by Gyimothy *et al.* [2]. However, they used such preprocessing to reduce metrics to the same interval before combining them for within-project prediction, whereas as explained, we use it to reduce the effect of project heterogeneity in cross-project prediction.

It is worth noting that the same models, applied without such a standardization process, or with different standardization formulae, produced worse results than those shown in Section V.

B. Multi-Objective Defect Prediction Models

One of the widely used machine learning technique is the multivariate logistic regression. Such a technique is particularly suitable to defect prediction, where there are only two possible outcomes: either the software component is defect-prone or it is not. In this paper we formulate the definition of a defect prediction model as a multi-objective problem, to deal with the competing nature of two needs: (i) the capability of the approach to identify defect-prone classes, and (ii) the cost of analyzing the predicted defect-prone classes. As said in the introduction, we choose not to consider precision and recall as the two contrasting objectives, because precision is less relevant than inspection cost—although proportional to it—when choosing the most suitable predictor. Similarly, it would not make sense to build models using cost and precision as objectives, as, again, precision captures the same dimension of cost, i.e., a low precision would imply a high code inspection cost.

Formally, let Y be a set of software components in the training set and let CP be the corresponding component-by-predictor matrix, i.e., an $m \times n$ matrix where m is the number of considered prediction metrics, and n is the number of source code components of a software system (e.g., classes). The generic entry $cp_{i,j}(k)$ of CP denotes the value of the i^{th} metric (used as predictor) for the j^{th} software component. The mathematical model used for regression has the form:

$$p(y_j) = \frac{e^{\alpha+\beta_1\ cp_{j,1}+\ldots+\beta_n\ cp_{j,n}}}{1+e^{\alpha+\beta_1\ cp_{j,1}+\ldots+\beta_n\ cp_{j,n}}} \quad (1)$$

where $p(y_j)$ is the estimated probability that the j^{th} software component is defect-prone, while the scalars $(\alpha, \beta_1, \ldots, \beta_n)$ represent the linear combination coefficients for the predictors $(cp_{j,1}, \ldots, cp_{j,n})$. Since $p(y_j)$ ranges in the interval $[0;1]$, $P(y_j)$ is usually set to 1 if $p(y_j) > 0.5$, 0 otherwise. Then, the proposed *multi-objective defect prediction problem* consists of finding a set of coefficients $c = (\alpha, \beta_1, \ldots, \beta_n)$

that (near) optimizes the following objective functions:

$$\max \text{ effectiveness}(c) = \sum_{i=1}^{m} P(y_i) \cdot B(y_i)$$

$$\min \text{ cost}(c) = \sum_{i=1}^{m} P(y_i) \cdot LOC(y_i)$$

where $P(y_i)$ represents the Boolean (i.e., 0 or 1) estimated defect-proneness of y_i, $B(y_i)$ is a Boolean value indicating the actual defect-proneness of y_i, and $LOC(y_i)$ measures the number of lines of code of y_i.

Since we have two different and contrasting objectives to be optimized, we use a definition of optimality based on Pareto dominance [16]:

Definition 1. *We said that a vector of coefficients c_i dominates another vector of coefficients c_j (also written $c_i \geq_p c_j$) if and only if the values of two objective functions (effectiveness and cost) satisfy:*

$$\textit{cost}(c_i) \leq \textit{cost}(c_j) \text{ and } \textit{effectiveness}(c_i) > \textit{effectiveness}(c_j)$$
$$\textit{or}$$
$$\textit{cost}(c_i) < \textit{cost}(c_j) \text{ and } \textit{effectiveness}(c_i) \geq \textit{effectiveness}(c_j)$$

Conceptually, the above definition means that c_i is better than c_j if and only if, at same level of effectiveness, c_i has a lower inspection cost than c_j. Alternatively, c_i is better than c_j if and only if, at same level of inspection cost, c_i has a greater effectiveness than c_j.

Starting from the above definition, the concept of optimality for the two-objectives problem is defined as [16]:

Definition 2. *A solution c_i is Pareto optimal if and only if it is not dominated by any other solution in the feasible region.*

In others words, a solution c_i is Pareto optimal if and only if no other solution c_j exists which would improve effectiveness, without worsening the inspection cost, and *vice versa*. All *decision vectors* (i.e., c_i) that are not dominated by any other decision vectors are said to form a *Pareto optimal set*, while the corresponding *objective vectors* (containing the values of two objective functions *effectiveness* and *cost*) are said to form a *Pareto frontier*. Identifying a Pareto frontier is particularly useful because the software engineer can use the frontier to make a well-informed decision that balances the trade-offs between the two objectives. In other words, the software engineer can choose the solution with lower inspection cost or higher effectiveness on the basis of the resources available for testing the predicted defect-prone components.

C. Training the Multi-Objective Predictor using Genetic Algorithms

Since the logistic regression model can be seen as a *fitness function* to be optimized, we use a multi-objective Genetic Algorithm (GAs) for dealing this problem. GA starts with an initial set of random solutions (random vectors of coefficients c_i in our case) called *population*, where each individual in the population is called *chromosome*. The population evolves through a series of iterations, called *generations*. To create the next generation, new individuals, called *offspring*, are formed by either merging two individuals in the current generation using a *crossover* operator or modifying a solution using a *mutation* operator. A new generation is formed with the *selection* of some parents and offspring, and rejecting others so as to keep the *population size* constant. The selection is based according to the values of the two objective functions (effectiveness and cost) of different individuals, thus fitter chromosomes have higher probability to survive in the next generations. After a sufficient amount of generations, the algorithm converge to "stable" solutions, which represent the optimal or near-optimal solutions to the problem.

Several variants of multi-objective GA have been proposed, each of which differs from the others on the basis of how the contrasting objective goals are combined for building the selection algorithm. In this work we used a multi-objective GA called vNSGA-II, proposed by Deb *et al.* [17].We applied the vNSGA-II for solving the multi-objective logistic regression problem on the training set, obtaining a set of non dominated solutions with different (and not furthermore improvable) effectiveness and cost values on the training set itself. After that, each Pareto optimal solution (that is a vector of coefficients for the software metrics) can be used for predicting defect-prone classes in the test set.

IV. Empirical Study Definition

This section describes the study we conducted aiming at evaluating the approach described in Section III. Specifically, the *goal* of the study was to evaluate the proposed multi-objective defect prediction approach, with the *purpose* of investigating the benefits introduced—in a cross-project context—by the proposed multi-objective prediction. The *quality focus* of the study is the cost-effectiveness of the proposed approach, compared to the performances of within-project prediction, single-objective prediction, and local, cross-project prediction proposed by Menzie *et al.* [10]. The *perspective* is of researchers aiming at developing a better cross-project prediction model, that could be used by software project managers especially in scenarios where the availability of project data does not allow a reliable within-project defect prediction.

The study context consists of 10 Java projects where information on defects is available. All of them come from the Promise repository[3]. A summary of the project characteristics is reported in Table I. The dataset report the actual defect-proneness of classes, plus a pool of metrics

[3]https://code.google.com/p/promisedata/

Table I
DATASETS USED IN THE STUDY.

Name	Release	Classes	Defect-prone Classes	(%)
Ant	1.7	745	166	22%
Camel	1.6	965	188	19%
Ivy	2.0	352	40	11%
Jedit	4.0	306	75	25%
Log4j	1.2	205	189	92%
Lucene	2.4	340	203	60%
Poi	3.0	442	281	64%
Prop	6.0	661	66	10%
Tomcat	6.0	858	77	9%
Xalan	2.7	910	898	99%

used as independent variables for the predictors, i.e., LOC and Chidamber & Kemerer Metrics.

A. Research Questions

This study aimed at addressing the following research questions:

- **RQ1**: *How does the multi-objective prediction perform, compared to single-objective prediction?*
- **RQ2**: *How does the proposed approach perform, compared to the local prediction approach by Menzie* et al. *[10]?*

In essence, **RQ1** evaluates the benefits introduced by the multi-objective prediction, whereas **RQ2** compares the proposed approach with a state-of-the-art approach—proposed by Menzie *et al.* [10]—that uses local prediction to mitigate the heterogeneity of projects in the context of cross-project defect prediction.

Overall, the experiments performed in our evaluation studied the effect of the following independent variables[4]:

- *Prediction technique:* in **RQ1** we compare a traditional, single-objective logistic regression with the multi-objective logistic regression. The former is a traditional model (available in R) in which the model is built by fitting data in the training set. The latter, as explained in Section III-B, is a set of Pareto optimal predictors built by a multi-objective GA, achieving different cost-effectiveness tradeoffs.
- *Local vs. global prediction:* we consider—within **RQ2**–both local prediction (using the clustering approach by Menzie *et al.* [10]) and global prediction.
- *Within-project vs. cross-project prediction*. The comparison of cross-project prediction and within-project prediction is done in both RQs.

In terms of dependent variables, we evaluated our models using (i) recall, which provides a measure of the model effectiveness; (ii) code inspection *cost*, measured as done by Rahman *at al.*[12] as the KLOC of the of classes predicted as defect-prone; and (iii) precision, which is an accuracy measure, which we report for the sake of comparison with other approaches.

[4]These are the study independent variables, not to be confounded with the prediction model independent variables.

It is important to note that we report precision and recall related to the prediction of defect-prone classes. It is advisable not to aggregate them with the prediction of non-defect-prone classes: a model with high (overall) precision (say 90%) when the number of defect-prone classes is very limited (say 5%), performs worse than a constant classifier. To compute precision and recall, we use a cross-validation procedure [18]. Specifically, for the within-project prediction, we used a 10-fold cross validation. For the cross-project prediction, we applied a similar procedure, removing each time a project from the set, training on 9 projects and predicting on the 10^{th} one.

Analyses for **RQ1** have been performed using *MATLAB Global Optimization Toolbox* (release R2011b). In particular, the *gamultiobj* routine was used for running the vNSGA-II algorithm, while the routine *gaoptimset* was used for setting the GA parameters. We used the GA configuration typically used for numerical problems [16]. Specifically:

- **Population size:** we choose a moderate population size with $p = 100$.
- **Initial population:** for each software system the initial population is uniformly and randomly generated within the solution spaces. Since such a problem in unconstrained, i.e. there are no upper and lower bounds for the values that the coefficients can assume, the initial population was randomly and uniformly generated in the interval $[-1000; 1000]$.
- **Number of generations:** we set the maximum number of generation equal to 400.
- **Crossover function:** we use arithmetic crossover with probability $p_c = 0.60$.
- **Mutation function:** we use a uniform mutation function with probability $p_m = 0.05$.
- **Stopping criterion:** if the average Pareto spread is lower than 10^{-8} in the subsequent 50 generations, then the execution of the GA is stopped.

The algorithm has been executed 30 times on each object program to account the inherent randomness of GAs. Then, we select a Pareto front composed of points achieving the median performance across the 30 runs.

B. Local Prediction Model by Menzie et al. [10]

To better assess the cross-project prediction accuracy of the proposed approach, we compared it with the local prediction approach proposed by Menzie *et al.* [10]. In the following we briefly explain the approach that was re-implemented using the *R* environment[5] (specifically, the *RWeka* and *cluster* packages). The prediction process consists of three steps: (i) data preprocessing, (ii) data clustering, and (iii) local prediction.

1) Data preprocessing: First, we preprocess the data set as described in Section III-A.

[5]www.r-project.org

2) Data Clustering: Second, we cluster together classes having similar characteristics. We use the k-means clustering algorithm available in the *R cluster* package, grouping together classes having similar values of the (preprocessed) metrics.

A critical factor in such a process is determining the most appropriate number of clusters for our prediction purpose, denoted as k in the k-means algorithm. To this aim, we use a widely-adopted approach from the cluster literature, the Silhouette coefficient [19]. The Silhouette coefficient is computed for each document to be clustered (class in our case) using the concept of cluster centroid. Formally, let C be a cluster; its centroid is equal to the mean vector of all classes c_i belonging to C: $Centroid(C) = \sum_{c_i \in C} c_i / |C|$. Starting from the definition of centroids, the computation of the Silhouette coefficient consists of the following three steps. First, for class c_i, we calculate the maximum distance from c_i to the other documents in its cluster. We call this value $a(c_i)$. Second, for class c_i, we compute the minimum distance from c_i to the centroids of the clusters not containing c_i. We call this value $b(c_i)$. Third, for class c_i, the Silhouette coefficient $s(c_i)$ is computed as:

$$s(c_i) = \frac{b(c_i) - a(c_i)}{\max\left(a(c_i), b(c_i)\right)}$$

The value of the Silhouette coefficient ranges between -1 and 1. A negative value is undesirable, because it corresponds to the case in which $a(c_i) > b(c_i)$, i.e., the maximum distance to other classes in the cluster is greater then the minimum distance to classes in other clusters. In essence, the (near) optimal number of clusters corresponds to the maximum value of the Silhouette, or to its elbow. Based on the Silhouette coefficient, in our study we found a number of clusters *k =10* to be optimal.

3) Local prediction: Finally, we perform a local prediction within each cluster. Basically, for each cluster obtained in the previous step, we use classes from $n - 1$ projects to train the model, and then we predict defects for classes of the remaining project. As predictor, we use the Logistic Regression implemented in the *R* package *RWeka*.

V. Study Results

This section discusses the results of our study aimed at answering the research questions stated in Section IV-A. Working data sets are available for replication purposes[6].

A. *RQ1: How does the multi-objective prediction perform, compared to single objective prediction?*

To assess the performances of the proposed approach, we first compare the prediction capabilities of the multi-objective logistic regression predictor with those of a single objective logistic regression. Also, we compare the cross-project prediction performances with the within-project prediction, again performed by a single-objective logistic regression.

Figure 1 shows the multi-objective cross-project predictor Pareto front, when predicting defects on each of the 10 projects after training the model on the remaining 9. Note that the lines in Figure 1 represent the median values across Pareto fronts obtained in the 30 GA runs. Also, the figure shows, as single points: (i) the accuracy of the single-objective cross-project logistic predictor (a cross "+"); (ii) the accuracy of the single-objective within project logistic predictor where the prediction has been performed using 10-fold cross validation within the same project (a star "*"). Finally, the graph also shows (as a triangle "Δ") the accuracy of the local prediction, clustering-based approach by Menzie *et al.* [10].

Table II reports a numerical comparison of the within-project prediction with the multi-objective cross-project prediction. Specifically, the table reports the cost and the precision of the two models for the same value of the recall, i.e., the value achieved by the single-objective model. Not surprisingly, the within-project model achieves, for 6 out of 10 projects, a better precision; the precision is the same for both models for *Log4j*, *Prop*, and *Xalan*. As for *Lucene* the cross-project prediction outperforms the within-project prediction. This result is consistent with results of previous studies [9] showing that in general within-project prediction outperforms cross-project prediction, due to the differences among the various projects.

However, although the precision decreases, we can notice how, for 7 projects, there is a cost decrease, in terms of KLOC to be analyzed, while for one project (*Xalan*) the cost remains the same. For two projects, *Camel* and *Poi*, the cost increases. The interpretation of such a decrement is straight-forward: while a lower precision for the same level of recall means analyzing more classes, this does not mean analyzing a larger amount of source code: on the contrary, for the same level of recall, and for 7 out of 10 projects, the multi-objective cross-project prediction model requires to analyze less code than the within-project predictor.

Table III compares results of the multi-objective cross-project predictor with those of the single-objective predictor, for the same level of recall. For 7 out of 10 projects (*Ivy*, *Jedit*, *Log4j*, *Lucene*, *Prop*, *Tomcat*, and *Xalan*) the multi-objective predictor allows to achieve the same level of recall with a lower cost in terms of KLOC. The precision decreases for two of these projects (*Jedit* and *Prop*), while for the others we can even observe a slight increase in terms of precision (*Log4j*, *Lucene*, and *Tomcat*), or the same value of precision (*Ivy* and *Xalan*). The only project for which the cost increases is *Poi*, where, however, the precision slightly increases. Finally, for Ant and Xalan both models achieve—for the same level of recall—the same performances in terms

[6]http://www.distat.unimol.it/reports/MO-DefectPrediction

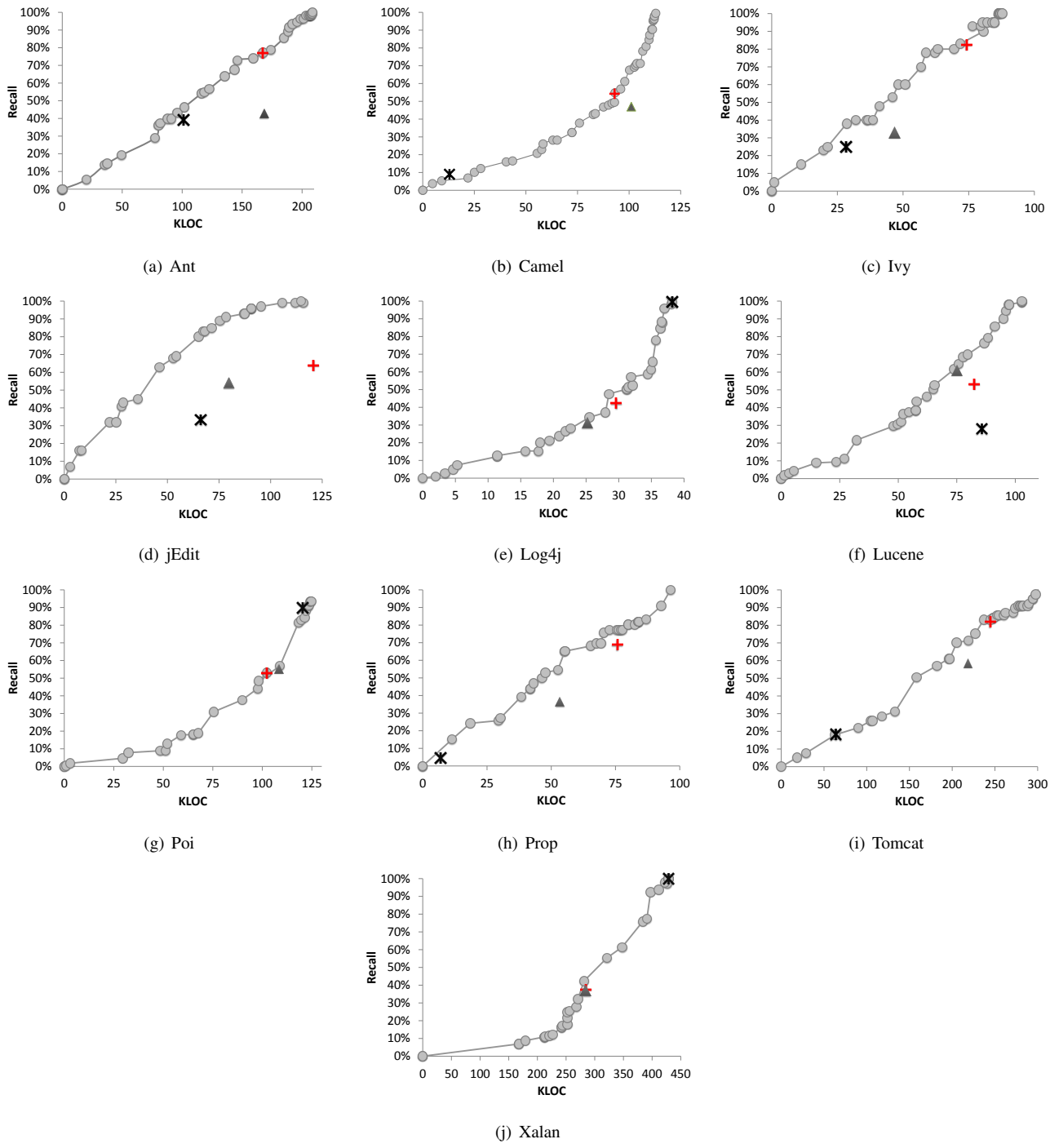

Figure 1. Comparison between Multi-objective Logistic and Single-objective Logistic.

Table II
MULTI-OBJECTIVE LOGISTIC VS. WITHIN PROJECT LOGISTIC (COST = KLOC TO ANALYZE, W = WITHIN PROJECT AND MO = MULTI-OBJECTIVE)

System	Metric	W	MO	Diff.
Ant	Cost	101	85	-16 (-16%)
	Rec	0.39	0.39	-
	Prec	0.68	0.51	-0.17
Camel	Cost	13	22	+9 (+69%)
	Rec	0.09	0.09	-
	Prec	0.54	0.40	-0.14
Ivy	Cost	28	21	-7 (-25%)
	Rec	0.25	0.25	-
	Prec	0.71	0.45	-0.26
Jedit	Cost	66	25	-41 (-62%)
	Rec	0.33	0.33	-
	Prec	0.66	0.13	-0.53
Log4j	Cost	38	37	-1 (-3%)
	Rec	0.99	0.99	-
	Prec	0.92	0.92	0
Lucene	Cost	86	45	-41 (-48%)
	Rec	0.28	0.28	-
	Prec	0.74	0.87	+0.13
Poi	Cost	120	122	+2 (-2%)
	Rec	0.90	0.90	-
	Prec	0.79	0.72	-0.07
Prop	Cost	7	6	-1 (-14%)
	Rec	0.05	0.05	-
	Prec	0.17	0.17	0
Tomcat	Cost	64	63	-1 (-2%)
	Rec	0.18	0.18	-
	Prec	0.58	0.31	-0.27
Xalan	Cost	429	429	0
	Rec	1	1	-
	Prec	0.99	0.99	0

Table III
MULTI-OBJECTIVE LOGISTIC VS. SINGLE-OBJECTIVE LOGISTIC (COST = KLOC TO ANALYZE, SO = SINGLE-OBJECTIVE LOGISTIC AND MO = MULTI-OBJECTIVE LOGISTIC)

System	Metric	SO	MO	Diff.
Ant	Cost	167	167	0
	Rec	0.77	0.77	-
	Prec	0.43	0.43	0
Camel	Cost	93	93	0
	Rec	0.54	0.54	-
	Prec	0.26	0.26	0
Ivy	Cost	74	72	-2 (-3%)
	Rec	0.83	0.83	-
	Prec	0.27	0.27	0
Jedit	Cost	121	43	-76 (-63%)
	Rec	0.64	0.64	-
	Prec	0.42	0.18	-0.24
Log4j	Cost	30	28	-2 (-7%)
	Rec	0.42	0.42	-
	Prec	0.94	0.95	+0.01
Lucene	Cost	83	53	-30 (-36%)
	Rec	0.53	0.53	-
	Prec	0.8	0.85	+0.05
Poi	Cost	102	103	+1 (1%)
	Rec	0.53	0.53	-
	Prec	0.87	0.88	+0.01
Prop	Cost	76	66	-10 (-13%)
	Rec	0.69	0.69	-
	Prec	0.67	0.17	-0.50
Tomcat	Cost	244	236	-8 (-3%)
	Rec	0.82	0.82	-
	Prec	0.21	0.23	+0.02
Xalan	Cost	285	276	-9 (-3%)
	Rec	0.38	0.38	-
	Prec	1	1	0

of cost and precision.

In summary, compared to the single-objective model, the multi-objective logistic regression predictor is able to achieve the same level of recall with lower cost, without even sacrificing precision (in most cases). Clearly, this is not the only advantage of the multi-objective model. Single-objective models try to achieve a compromise between precision and recall. This for some projects—such as *Ivy* or *Tomcat*—means a high recall and a low precision, whereas for others—such as *Log4j* or *Xalan*—it means a relatively low recall and a high precision. Multi-objective models, instead, allow the software engineer to understand, based on the amount of code she can analyze, the level of recall that can be achieved, and therefore to select the most suited predictors. For example, for *Log4j* it would be possible to achieve a recall of 80% instead of 37% if analyzing 35 KLOC instead of 28 KLOC. For *Xalan*, achieving a recall of 80% instead of 38% means analyzing about 400 KLOC instead of 276 KLOC. In this case, the higher additional cost can be explained because most of the *Xalan* classes are defect-prone; therefore achieving a good recall means analyzing most of the system, if not the entire system. Let us suppose, instead, to have only a limited time available to perform our analysis/testing tasks. For *Ivy*, we could decrease the cost from 72 KLOC to 50 KLOC if we accept a recall of 50% instead of 83%. For *Tomcat*, we can decrease the cost from 236 to 150 KLOC accepting a recall of 50% instead of 82%.

To provide statistical evidence to the achieved results, we have performed a Wilcoxon two-tailed test to compare the cost difference (in percentage) between the single-objective and the multi-objective predictor, and the difference in terms of precision. Assuming a significance level of 5%, results indicate that, while there is no significant difference in terms of precision (p-value=0.40), the cost is significantly lower (p-value=0.02).

RQ1 Summary. In agreement with Zimmermann *et al.*, [9], the cross-project predictor achieves a lower precision than within project prediction. However, as also found by Rahman *et al.* [12], for the same level of recall, in most cases the multi-objective model is able to achieve a lower analysis cost.

*B. **RQ2**: How does the proposed approach perform, compared to the local prediction approach by Menzie* et al.*?*

In this section we compare the accuracy of the cross-project multi-objective logistic predictor with an alternative method for cross-project predictor, i.e., the "local" predictor based on clustering developed by Menzie *et al.* [10]. Table IV shows cost and precision—for both the local prediction approach and the multi-objective approach—for the level of recall achieved by the local prediction approach. As the Table shows, for the same level of recall, the multi-objective approach always achieves a lower cost, with the

Table IV
MULTI-OBJECTIVE LOGISTIC VS. CLUSTERING BASED LOGISTIC (COST = KLOC TO ANALYZE, CL = CLUSTERING AND MO = MULTI-OBJECTIVE)

System	Metric	CL	MO	Diff.
Ant	Cost	168	96	-72 (-43%)
	Rec	0.43	0.43	-
	Prec	0.20	0.43	+0.23
Camel	Cost	101	88	-13 (-13%)
	Rec	0.47	0.47	-
	Prec	0.23	0.27	+0.04
Ivy	Cost	47	25	-23 (-49%)
	Rec	0.33	0.33	-
	Prec	0.11	0.44	+0.33
Jedit	Cost	80	39	-41 (-51%)
	Rec	0.54	0.54	-
	Prec	0.24	0.18	-0.06
Log4j	Cost	22	22	0
	Rec	0.31	0.31	-
	Prec	0.95	0.98	+0.03
Lucene	Cost	75	73	-2 (-3%)
	Rec	0.61	0.61	-
	Prec	0.65	0.69	+0.04
Poi	Cost	108	103	-5 (-5%)
	Rec	0.55	0.55	-
	Prec	0.72	0.88	+0.016
Prop	Cost	53	35	-18 (-34%)
	Rec	0.36	0.36	-
	Prec	0.10	0.20	+0.10
Tomcat	Cost	219	182	-37 (-17%)
	Rec	0.58	0.58	-
	Prec	0.12	0.30	+0.18
Xalan	Cost	284	275	-9 (-3%)
	Rec	0.37	0.37	-
	Prec	1	1	0

exception of *Log4j*, for which the cost is the same while the recall is slightly better. The multi-objective approach also achieves a better precision, except for *Jedit* where it decrease of 6%, with a cost decrease of 41 KLOC, and for *Xalan* where it remains the same. If we focus on the two projects also used by Menzie *et al.*—*Lucene* and *Xalan*—we notice that the multi-objective approach achieves a cost reduction of 2 KLOC and 9 LKOC respectively, with a precision increase of 4% for *Lucene*, and same precision for *Xalan*.

Therefore, results clearly indicate that the multi-objective approach outperforms the local prediction approach. The Wilcoxon test indicates, for a significance level of 5%, that differences are also statistically significant (p-value=0.009) for what concerns the cost, and marginally significant (p-value=0.05) for what concerns the precision. In addition to that, and as also discussed for **RQ1**, the multi-objective model provides more flexibility to the software engineer, allowing to choose predictors with high cost and recall, or predictors with lower cost and lower recall.

RQ2 Summary. For the analyzed projects, the multi-objective cross-project predictor outperforms the local predictor by Menzie *et al.* [10] achieving, for the same level of recall, a significantly lower cost, and a marginally significant higher precision.

VI. THREATS TO VALIDITY

This section discusses the threats to validity that can affect the evaluation of the proposed approach and of the reported study.

Threats to *construct validity* concern the relation between theory and experimentation. Some of the measures we used to assess the models (precision and recall) are widely adopted ones. In addition, we use the amount of LOC to be analyzed as a proxy indicator of the analysis/testing cost, as also done by Rahman *et al.* [12]. We are aware that such a measure is not necessarily representative of the testing cost especially when black-box testing techniques or object-oriented (e.g., state-based) testing techniques are used. Also, another threat to construct validity can be related to the used metrics and defect data sets. Although we have performed our study on widely used data sets from the Promise repository, they can be prone to imprecision and incompleteness.

Threats to *internal validity* concern factors that could influence our results. We mitigated the influence of the GA randomness when building the model by repeating the process 30 times and reporting the median values achieved. We also investigate whether the performance of our approach depends on the particular machine learning technique being used (logistic regression). We also tried ADTree, obtaining consistent results. Future work will aim at experimenting further techniques.

Threats to *conclusion validity* concern the relationship between the treatment and the outcome. In addition to showing values of cost, precision and recall, we have also statistically compared the various model using the Wilcoxon, non-parametric test, indicating whether differences in terms of cost and precision are statistically significant.

Threats to *external validity* concern the generalization of our findings. Although we considered data from 10 projects, the study deserves to be replicated on further projects, for example projects developed using different programming languages (we considered Java projects only). Also, we only considered defect prediction models based on some specific sets of product metrics, e.g., Chidamber & Kemerer metrics and LOC. We do not know whether the proposed approach would produce similar accuracy if applied to process metrics or to other kinds of product metrics.

VII. CONCLUSION AND FUTURE WORK

This paper proposed a novel, multi-objective defect prediction approach based on a logistic regression model trained by means of a vNSGA-II [17] Genetic Algorithm. The goal is to produce, instead of a single defect prediction model—achieving a compromise between precision and recall—a Pareto front of predictors that allow to achieve different tradeoff between the cost of code inspection—measured in this paper in terms of KLOC of the source code artifacts (class)—and the amount of defect-prone classes that the model can predict (i.e., recall). As recently pointed out by Rahman *at al.* [12], cost-effectiveness is a more meaningful indicator of the practical usefulness of a defect prediction

model, and, in such terms, cross-project defect prediction does not perform worse than within-project prediction.

We applied the proposed approach to perform cross-project defect prediction across 10 Java projects. Results indicate that:

1) cross-project prediction is, as also found by Zimmermann *et al.* [9] worse than within-project prediction in terms of precision and recall; however
2) the proposed multi-objective model allows to achieve a better cost-effectiveness than within-project predictors, and better than single-objective predictors;
3) the multi-objective predictor outperforms a state-of-the-art approach for cross-project defect prediction by Menzie *et al.* [10], based on local prediction among classes having similar characteristics;
4) finally, the multi-objective approach, instead of producing a single predictor allows the software engineer to choose the configuration that better fits her needs, in terms of recall and of amount of code she can inspect. In other words, the multi-objective model is able to tell the software engineer how much code one needs to analyze to achieve a given level of recall.

In summary, the proposed approach seems to be particularly suited for cross-project defect prediction, although the advantages of such a multi-objective approaches can also be exploited in within-project predictions.

Future work aims at considering different kinds of cost-effectiveness models. As said, LOC is a proxy for code inspection cost, but certainly is not a perfect indicator of the cost of analysis and testing. Alternative cost models, better reflecting the cost of some testing strategies (e.g., code cyclomatic complexity for white box testing, or input characteristics for black box testing) must be considered. Also, in terms of effectiveness, we plan to consider the number of defects covered by the prediction rather than the number of defect-prone classes. Last, but not least, we plan to investigate whether the proposed approach could be used in combination with—rather than as an alternative—to the local prediction by Menzie *et al.* [10].

REFERENCES

[1] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Software Eng.*, vol. 22, no. 10, pp. 751–761, 1996.

[2] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction." *IEEE Trans. Software Eng.*, vol. 31, no. 10, pp. 897–910, 2005.

[3] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Software Eng.*, vol. 20, no. 6, pp. 476–493, 1994.

[4] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *30th International Conference on Software Engineering (ICSE 2008)*. Leipzig, Germany, May: ACM, 2008, pp. 181–190.

[5] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Trans. Software Eng.*, vol. 31, no. 4, pp. 340–355, 2005.

[6] S. Kim, T. Zimmermann, J. E. Whitehead, and A. Zeller, "Predicting faults from cached history," in *29th International Conference on Software Engineering (ICSE 2007)*. Minneapolis, MN, USA, May 20-26, 2007: IEEE Computer Society, 2007, pp. 489–498.

[7] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *28th International Conference on Software Engineering (ICSE 2006)*. Shanghai, China, May 20-28, 2006: ACM, 2006, pp. 452–461.

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

[9] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process," in *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2009, pp. 91–100.

[10] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. R. Cok, "Local vs. global models for effort estimation and defect prediction," in *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), 2011*. IEEE, 2011, pp. 343–351.

[11] N. Bettenburg, M. Nagappan, and A. E. Hassan, "Think locally, act globally: Improving defect and effort prediction models," in *9th IEEE Working Conference o Mining Software Repositories, MSR 2012, June 2-3, 2012, Zurich, Switzerland*. IEEE, 2012, pp. 60–69.

[12] F. Rahman, D. Posnett, and P. Devanbu, "Recalling the "imprecision" of cross-project defect prediction," in *Proceedings of the ACM-Sigsoft 20th International Symposium on the Foundations of Software Engineering (FSE-20)*. Research Triangle Park, NC, USA: ACM, 2012, p. 61.

[13] Y. Collette and P. Siarry, *Multiobjective Optimization: Principles and Case Studies*. Springer, 2004.

[14] L. C. Briand, W. L. Melo, and J. Würst, "Assessing the applicability of fault-proneness models across object-oriented software projects," *IEEE Transactions on Software Engineering*, vol. 28, pp. 706–720, 2002.

[15] A. E. Camargo Cruz and K. Ochimizu, "Towards logistic regression models for predicting fault-prone code across software projects," in *Proceedings of the Third International Symposium on Empirical Software Engineering and Measurement (ESEM 2009)*, Lake Buena Vista, Florida, USA, 2009, pp. 460–463.

[16] C. A. Coello Coello, G. B. Lamont, and D. A. V. Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast elitist multi-objective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, 2000.

[18] M. Stone, "Cross-validatory choice and assesment of statistical predictions (with discussion)," *Journal of the Royal Statistical Society B*, vol. 36, pp. 111–147, 1974.

[19] J. Kogan, *Introduction to Clustering Large and High-Dimensional Data*. New York, NY, USA: Cambridge University Press, 2007.