EkmEx - An Extended Framework for Labeling an Unlabeled Fault Dataset

Muhammad Rizwan
Department of Computer Science
Capital University of Science and Technology, Islamabad, Pakistan

Aamer Nadeem
Department of Computer Science
Capital University of Science and Technology, Islamabad, Pakistan

Sohail Sarwar
Department of Computer Science
London South Bank University
England

Muddesar Iqbal
Department of Computer Science
London South Bank University
Engaland

Muhammad Safyan
Department of Computer Science
GC University, Lahore
Pakistan

Zia Ul Qayyum
Department of Computer Science
Allama Iqbal Open University
Pakistan

***Abstract*—Software fault prediction (SFP) is a quality assurance process that identifies if certain modules are fault-prone (*FP*) or not-fault-prone (*NFP*). Hence, it minimizes the testing efforts incurred in terms of cost and time. Supervised machine learning techniques have capacity to spot-out the *FP* modules. However, such techniques require fault information from previous versions of software product. Such information, accumulated over the life-cycle of software, may neither be readily available nor reliable. Currently, clustering with experts' opinions is a prudent choice for labeling the modules without any fault information. However, the asserted technique may not fully comprehend important aspects such as selection of experts, conflict in expert opinions, catering the diverse expertise of domain experts etc. In this paper, we propose a comprehensive framework namedEkmEx that extends the conventional fault prediction approaches while providing mathematical foundation through aspects not addressed so far. The EkmEx guides in selection of experts, furnishes an objective solution for resolve of verdict-conflicts and manages the problem of diversity in expertise of domain experts. We performed expert-assisted module labeling through EkmEx and conventional clustering on seven public datasets of NASA. The empirical outcomes of research exhibit significant potential of the proposed framework in identifying *FP* modules across all seven datasets.**

***Index Terms*—Expert opinion, Labeling datasets, Software fault proneness, Software metrics**

I. Introduction

A fault is an error that causes non-conformity [1] with software requirements. In software systems, fault is an unavoidable problem incurred by internal defects. Software faults can cause huge losses and catastrophies. For example, Mariner 1 destroyed due to code error [2]. In 2003, the blackouts of the North-Eastern United States were also caused by software fault [3]. Software fault is closely related to security, reliability and maintainability of thesystem. In software projects, it is quite difficult for testers to find all the software faults. The program debugging process has been employed for fault identification in software systems. However, debugging is a costly activity in terms of time and effort exerted [4]. In recent years, complexity of software programs has increased manifold in terms of lines of code. Such complexity in software systems has made the software faults almost unavoidable. This gives rise to a major concern i.e. fault identifcation consumes nearly 80% of the total budget of a software project. These costs can be diminished if faults are identified earlier in the development process. Therefore, researchers are focusing on software fault prediction activities for estimating the number and distribution of faults.

Software fault prediction is a discipline that predicts fault proneness of certain modules using certain metrics and historic fault data. Software fault prediction refines the process of testing while identifying modules to be refactored in maintenance phase. Hence, it reduces the effort and time required to review the code [5–8].

The research community has proposed numerous fault prediction approaches, such as supervised learning models, unsupervised models, transfer learning models, and supervised learning/ active learning. These approaches help to determine the fault-prone *FP* and not fault-prone *NFP* modules.

SFP works on data of metrics that elaborate the description of software under test. The description can be related to design, code, complexity, or volume of the software artifact. These metrics can be at method level, class level, or even file-level. However, the metrics that capture the method-level details are dominant in use (60%) followed by class level metrics, which is 24% [9]. The comprehensive discussion on software metrics can be found in [10, 11]. According to the studies, [9, 12–18], Halstead [19], LOC, and McCabe cyclometic complexity [20] metrics are the most frequently used metric in non Object Oriented (OO) paradigm. Whereas in OO paradigm, Catal Systematic (CK) metric suite is used most frequently[9, 12–18].

The complexity-based software metrics reflect the cognitive complexity which is the main cause of fault [21–23] as shown in figure 1. The second important requirement of SFP approaches is the availability of labeled data. The label comprises of fault information, that could be binary such as *FP* or *NFP*, ordinal as the severity of faults, or a numerical that shows the number of faults in the software modules.

SFP research community is more inclined towards classifying *FP* and *NFP* modules [24, 25]. Unfortunately, such fault information is not always available, because of two main reasons: Firstly, similar software are not developed all the time; secondly, there is lack of fault tracking systems in similar software. Training the SFP model in such scenarios is quite challenging, which is addressed by many studies [26–28].

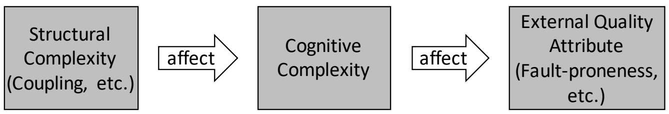

Fig. 1. Association of coupling complexity with software faults

Common approaches to resolve the stated challenges are Cross-product defect prediction (CPDP) [26, 29, 30], Bad Smell based defect labeling[31, 32], Clustering with expert opinion[33], Heuristic-based approaches[34–37], Rule-based approaches, and Threshold-based approaches[38, 39].
In this paper, we proposed a fault labeling framework named EkmEx that extends the conventional approaches i.e. clustering with expert opinion [33, 40]. The base technique uses a clustering technique for making two clusters of all the unlabeled modules. Later, the centroid of each cluster is presented to domain experts for labelling each of them as *FP* or *NFP*. Instead of analysing and labeling each module one by one, experts examine only the centroid and assign a label to all data points within same cluster. A major contribution of current research is to present a more systematic, reliable and robust approach in identifying *FP* modules. A comparison has been performed on seven public datasets of NASA to measure the effectiveness of proposed approach.
The rest of the paper is organized as follows: *Section II* presents the literature review of current research. The proposed framework has been discussed in *Section III* wherein the process of dataset development is elaborated followed by *Section IV* that compares the proposed framework with the existing one. Threats to the design of an experiment and results are stated in *Section V*. We conclude the paper and provide possible future directions in *Section VI*.

II. Literature Review

In the supervised software fault prediction, labeling is a challenging task. Information about faults is generally stored in the software archives. The task of associating fault to the modules is challenging even in the presence of the archives. Kim Herzig and Sascha [41] analyzed five open-source projects and reported that about 40% of reports are inaccurately classified. Consequently, about 40% of modules that were classified as *FP*, never had a bug. Likewise, Bird *et al.* analyzed 24 thousand fixed bugs, wherein only 10 thousand could be accurately linked to the bug dataset[42]. The reason for such discrepancy is that bug archives do not explicitly state which version the patch file is applied. Likewise, sometimes bugs are associated with classes that are generated at run-time. This would lead to the erroneous association of the bugs to static classes. This is the reason research community moved to other techniques for labeling the dataset [26, 43, 44]. This section briefly discussed these techniques.
CPDP is a Cross product defect prediction approach discussed by many researchers [26–28, 43, 44]. In this approach, the machine learning (ML) model is trained on the labeled dataset, which is then used to predict the label of unlabeled dataset. However, varying distributions between training and test dataset would not make it useful. Efforts have been made on solutions to improve the predictive performance of CPDP models, like transfer learning solution [45], data transformation [28], relevant instances filtration [46], etc. Still, a lot of work yet to be done to make it practically viable. Zhang et al. [47] employ Cross version Data Selection (CDS) technique to resolve the version selection problem. The proposed CDS framework applied clustering to existing files for detecting the similar and most noisy instances for training while applying the Weighted Sampling Model (WSM) to the new files. The results are quite appreciable by applying the framework to 28 versions across 8 software projects. Amaski et al. [48] suggested that the latest prior release is the better choice for the selection of training instances. The authors claim that CPDP approaches could improve CVDP (Cross-version defect prediction). Likewise, the project's latest release has an imperative advantage of usage as compared to the older releases.
Bad smell based approach was first used by Beck to describe 22 particular structures in code that have a negative effect on software and should be avoided.These rules originated from human expertise by interpreting information [31] [49]. Human experts usually convert the problem into a set of condition action rules which usually constituteif-then. In addition to that Metadata, metrics, and other necessary projects artifact-related information is fed into the rules. Sometimes such rules are automated to solve the process. Hall et al. [32] reported that coding rule violations have a small but significant effect on the occurrence of faults at file level. Likewise, Li and Shatnawi [50] studied the relationship of six smells with faults. Olbrich *et al.* [51, 52] studied three smells in the software system. Whereas, Sjoberg et al. [53] focused on the size of these effects, hence reported the effect of 12 smells. Gupta and Singh [54] employ metrics and rules to demonstrate the effectiveness of code smell in JAVA code. The aim was to detect temporary field code smell based on the selected metrics. The selected metrics are computed through data and control flow graphs. The empirical results advocate the effectiveness of temporarily filed code smell detection.
Clustering with experts' opinion develop clusters out of data and declare partial data as *FP* or *NFP*. Clusters are developed through expert-based approaches or non-expert based approaches. The base assumption is that modules that are coherent in their structure are likely to share the quality parameters (i.e. fault-proneness). Zhong *et al.* [33] used K-means [55] and Neural-Gas [56] clustering methods to cluster

hundreds of software modules into a small number of coherent groups. Subsequently, essential data statistics from each cluster were presented to experts having 15 years of experience in software engineering. The experts labelled the representative instances as *FP* or *NFP* based on their domain knowledge. However, the quality parameters were not resented to the experts. Finally, the labels of representative modules were assigned to all the data points of corresponding cluster. Following a similar pattern Seliya *et al.* [40] performed clustering of datasets from software projects by NASA through k-means and labelled the centroids through domain experts.

Rodriguez *et al.* [57] exploited two algorithms, the SD algorithm, and the CN2-SD algorithm to generate rules to determine the *FP* modules. They evaluated the generated rules on three datasets from NASA projects [58] and DAmbros *et al.* repository [59]. They use object-oriented metrics for generating clusters and identifying rules. Yuan *et al.* [60] introduced Fuzzy clustering, which was a regression model, to estimate the number of faults. The authors illustrate the method on large scale telecommunication systems and reported significant outperformance. The most recent work is performed by Yang *et. al* [37]. The authors propose a cluster ensembles labeling approach (CEL) and perform an experiment on 15 different datasets from three different repositories. CEL first makes multiple sub-optimal clusters and then combines into a single better cluster. The clustering and labeling are done through Access Control Lists (ACL) [61].

Heuristic based approach uses some cognitive logics to determine the *FP* instance. Nam and Kim [34] propose CLA/CLAMI, which neither requires prior fault data nor is influenced by the varying distribution of data. It is actually composed of two techniques CLA and CLAMI. The first two steps of both techniques are clustering modules and labeling modules in clusters. The technique named CLAMI has two more steps to overcome mislabeling i.e. Metric selection and Instance selection. The techniques are evaluated on seven open-source datasets and produced significantly viable results. However, the technique drops a significant number of instances and metrics, thus makes it nearly useless. Another heuristic-based approach is ACL, proposed by Yang *et al.* [35] in which labeling is based upon certain beliefs supported by some other researchers. Like, highly distributed change is more likely to be a defect inducing change. Likewise, larger change has a higher likelihood of being a defect-inducing change. The approach is four steps; the first step is getting the average of each metric. In the second step, for a certain instance, if the value of metric is greater than the half average of that very metric then the instance is preliminarily labeled as *FP*. Based on the violation score (MIVS) instances are further pruned in the last two steps. Andrew *et al.* [36] labeling is based upon the heuristic that "For most metrics, software entities containing defects generally have larger values than software entities without defects.". They took three datasets with 26 projects in total and proved that the connection between faulty and clean is smaller than the connection between faulty instances and the connection between clean instances. Yan *et al.* [62] used both supervised and unsupervised learning approaches on file level defect prediction and conclude that within the project later is not better however, across the project later is better. Labeling is done based on the experiment of [35].

Threshold based approaches: B. Ralf [38] take threshold through experience [63] and hints from literature, tuning machine, and analysis of multiple versions. Catal *et al.* [39] make clusters using X-means and then computes the mean vector of each cluster which is then compared with the thresholds vector. The complete cluster is predicted as *FP* if at least one metric of the mean vector is higher than the threshold value of that metric. X-means need k_{min} and k_{max} value. It uses a posterior probability to select the right number of clusters. The threshold for different metrics are;

1) LOC=65
2) v(g) =10
3) n1 =25
4) n2 =40
5) N1 =125
6) N2 =70

Values are taken from Integrated software metrics (ISM). Currently, the website is not working.

Non-clustering-based approaches have very narrow coverage due to a limited metrics' coverage. Similarly, they lack in accommodating newly developed software metrics. The approaches wherein clustering techniques are employed without expert opinion face the limitations as mentioned by [37, 39, 57, 60, 64, 65]. The reasons for shortcomings are static thresholds that have been predefined or non-availability of a labeled dataset for computing thresholds. On the other hand, clustering based approaches have no such limitations for considering expert opinions. Conclusively stating, these techniques hold following disadvantages:

1) Ignores clusters' quality assessment
2) Depends upon experts' availability
3) Ignores diverse experts' expertise
4) Ignores difference in experts' opinion
5) Does not compute the labelings' reliability

We tried to improve this approach while resolving the associated limitation by proposing a new framework EkmEx, that is elaborated in the forthcoming section.

III. EkmEx - An extended framework for fault labeling

EkmEx, **E**xtended **k-m**eans with **Ex**perts' opinion extends the conventional approach i.e. clustering with experts opinion [33]. The base technique makes two clusters of the un-labeled dataset that comprises of complexity metrics. After that, the centroid of each cluster is presented to the expert, who labels each centroid as *FP* or *NFP*. Subsequently, the given label is assigned to all the data points of the corresponding cluster. As discussed in prior sections, conventional approaches may not cater some important aspects. These aspects have been well-addressed by proposed framework i.e. EkmEx. In this section, proposed framework has been applied to seven datasets used in NASA projects.

A. Selected datasets

As a case study, seven public datasets by NASA [58] have been selected. Originally, NASA datasets consisted of 13 software projects in PROMISE version and 14 software projects in MDP version. There was a considerable difference in the number of feature-sets of both the projects[66]. The datasets had few quality problems [66]. Still, these datasets are used most widely by SFP community [67]. Table I shows a brief description and fault ratio in the selected datasets.

TABLE I
BRIEF DESCRIPTION OF SEVEN DATASETS BY NASA

Dataset	*n*	*NFP*	*FP*	*%FP*	Description
CM1	498	449	49	9.83	NASA spacecraft instrument written in "C"
KC1	2109	1783	326	15.45	KC1 is a "C++" system implementing storage management for receiving and processing ground data
KC2	522	415	107	20.49	Data from C++ functions. Science data processing; another part of the same project as KC1; different personnel.
KC3	458	415	43	9.39	Program written in Java
MC2	161	109	52	32.29	Program written in C++
MW1	434	403	31	7.14	Program written in C++
PC1	1109	1032	77	6.94	Data from C functions. flight software for earth orbiting satellit

We selected these datasets for three main reasons

1) These versions are widelyused in several empirical studies on SFP.
2) The datasets arepublicly and freely available which makes it easy for external validation of our empirical results by other researchers.
3) The datasets comprise the information of fault along with numerous software complexity metrics.
4) The datasets allow the label to be binary by just labeling *FP* for the erroneous module, otherwise *NFP*. Since the binary labels are the most used category by SFP community [24, 25].

Table II shows a brief description of the metrics in the selected datasets.

TABLE II
DESCRIPTION OF THE METRICS IN THE SEVEN DATASETS

Class of metric	Symbol	Description
McCabe[20, 68]	loc	McCabes Lines of code
	v(g)	Cyclomatic complexity
	eV(g)	Essential complexity
	iv(g)	Design complexity
Halstead [19]	n1	Unique Operators
	n2	uniqOperands
	N1	totalOperators
	N2	Total Operands
Branch	Br. Cnt	Total number of branches of the flow graph
Label	1	Presence of fault
	0	Absence of fault

The label shall act as a testing of our framework and comparison with the base technique. The brief statistical description of each metric in all selected datasets is shown in Table III.

TABLE III
STATISTICAL DESCRIPTION OF THE SELECTED SEVEN DATASETS

Dataset	Parameter	*loc*	*v(g)*	iv(g)	*ev(g)*	*n1*	*n2*	*N1*	*N2*	*br.Cnt*
CM1	mean	46.8	7.3	5	3.1	19.1	35.4	120.2	77	13
	std	56.2	9.5	6.4	4.2	9.3	38.2	151	98.5	16.8
	min	7	2	1	1	4	3	7	4	3
	25%	18	3	2	1	13	13	38	22	5
	50%	28	4	3	1	17	22	68	40	7
	75%	53	8	6	4	23	42	136.5	92.5	15
	max	503	96	63	30	72	314	1229	798	162
KC1	mean	46.5	8	7.1	4.6	13.9	24.8	98.6	59.4	15
	std	51.1	8.4	7.5	5.2	8	21.8	109.7	66.7	16.9
	min	0	1	1	1	0	0	0	0	1
	25%	3	1	1	1	7	4	11	5	1
	50%	33	5	4	1	15	22	71	44	9
	75%	72	12	11	7	20	36	136	84	23
	max	262	45	45	26	37	120	678	428	89
KC2	mean	36.9	4.9	3.7	2.4	9.2	14.5	57.6	37	8.8
	std	77.9	11	8.1	6.7	6.4	22.1	143	90.4	21.9
	min	1	1	1	1	1	0	1	0	1
	25%	4	1	1	1	3	2	4	2	1
	50%	13	2	2	1	8	7	16.5	11	3
	75%	45	5	4	1	14	20	64	41	9
	max	1275	180	143	125	47	325	2469	1513	361
KC3	mean	32.6	6.4	5.5	2.7	16.1	29.3	117.8	69	10.8
	std	34.2	5.6	4.8	3	4.9	25.5	126.7	78	10.1
	min	4	2	1	1	7	4	13	5	3
	25%	11.3	2	2	1	13	13	41.3	22	3
	50%	20	4	4	1	16	20	69.5	39	7
	75%	43	7.8	6.8	3	19	40.5	151.3	88.5	13
	max	242	36	34	20	31	159	857	556	59
MC2	mean	44	8.9	3.1	4.4	15.6	24	118.3	86.9	16.5
	std	52.7	11	3.7	7.4	6.6	21	147.3	104.5	21.7
	min	2	2	1	1	3	1	3	1	3
	25%	12	3	1	1	11	10	27	20	5
	50%	27	5	2	1	15	17	66	50	9
	75%	57	10	3	4	20	33	156	120	19
	max	306	63	21	39	32	98	864	615	125
PC1	mean	31	7.7	4.4	3.6	16.5	27	89.9	68.9	13.8
	std	41.3	10.5	7.8	6.8	8	33.7	130.6	99.5	19.3
	min	0	1	1	1	4	2	6	3	3
	25%	11	3	2	1	11	10	25	18	5
	50%	19	5	3	1	15	18	49	38	9
	75%	36	8	5	5	20	35	102	77	15
	max	602	136	123	123	99	538	1641	1144	236
MW1	mean	26.8	6	4.5	2.5	13.5	25.8	59.3	46.7	10.3
	std	19.8	5.2	4.1	3.2	6.5	17.2	48.7	38.4	9
	min	4	2	1	1	4	3	6	4	3
	25%	12	3	2	1	9	13	25	19	5
	50%	20	4	3	1	12	21	43	33	7
	75%	37	7	5	3	17	36	80	61	13
	max	112	28	26	23	44	94	303	226	50

B. Labeling through EkmEx

This subsection explains the proposed framework, EkmEx. We apply all the steps of EkmEx on the seven datasets shortlisted along with the elaboration of each step. The block diagram of EkmEx is shown in figure 2.

1) Clustering through k-means: In the first step, we made two clusters in all three selected datasets using k-means clustering algorithm [55]. It is an optimization algorithm that aims to minimize the mean-squared error. The k-means aims to split data instances into k clusters wherein each instance belongs to the cluster with the least mean-squared error. We selected this algorithm because of following two main reasons; (a) We can set the number of required clusters, which is two in our case i.e. *FP* and

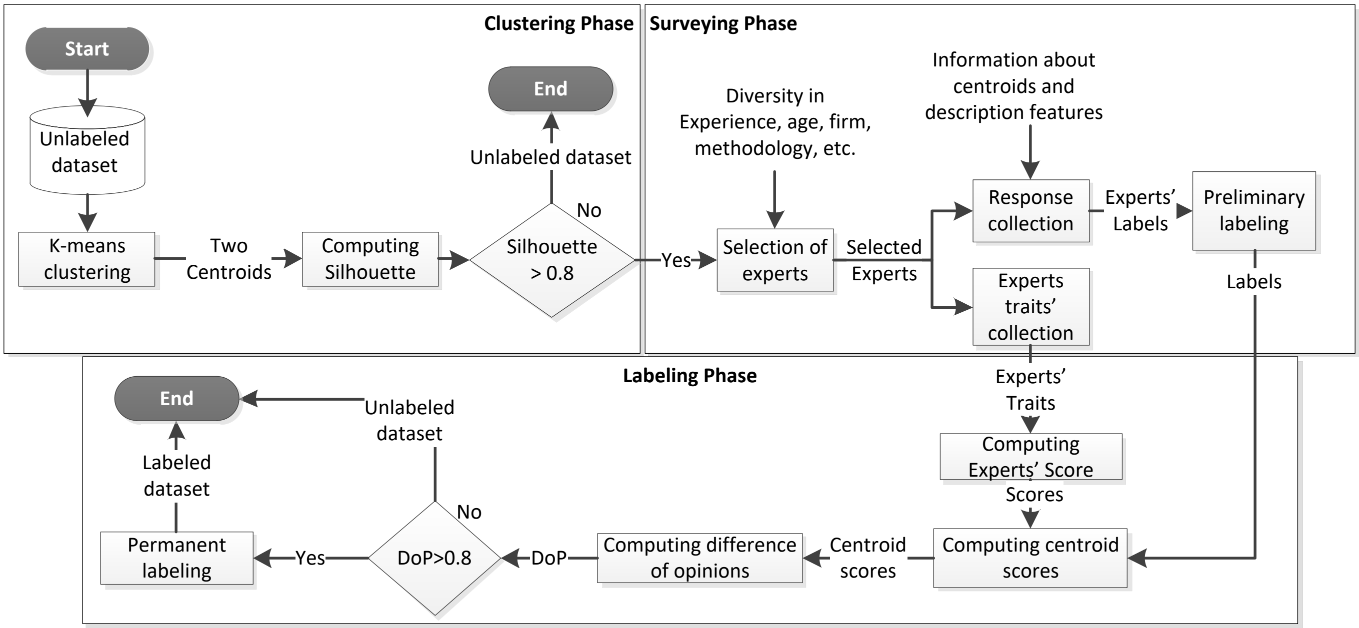

Fig. 2. Detailed EkmEx labeling framework

NFP. (b)We can easily select the most candidate instance that represents the whole cluster, which is the centroid in k-means.

TABLE IV
DETAIL OF CENTROIDS IN SEVEN DATASETS WITH SILHOUETTE SCORE

Datasets	*loc*	*v(g)*	*iv(g)*	*ev(g)*	*n1*	*n2*	*N1*	*N2*	*br.Cnt*	*Silhouette*
CM1	31.1	5	3.4	2.2	16.8	24.2	74	46.3	8.7	0.85
	150.6	22.8	15.3	8.8	34.1	109.8	425	279.4	41.5	
KC1	128.2	21.1	18.3	12.2	23.6	58.9	279.9	170.1	41.4	0.87
	27.0	4.9	4.4	2.7	110.5	16.7	55.3	32.9	8.7	
KC2	31.8	4.1	3.1	2	9	13.2	48.2	31	7.3	0.94
	699.5	103	75.3	61	33.3	176.8	1276.5	815.3	202.8	
KC3	90.9	15.2	12.9	6.1	22.7	72.8	338.7	204.5	26.5	0.70
	20.6	4.5	3.9	2	14.7	20.4	72.5	41.2	7.6	
MC2	33.2	6.7	2.7	2.8	14.8	21.4	87.3	65.8	12	0.87
	200.9	41.4	8.8	27.6	27.3	61.6	571.8	394.6	81.6	
MW1	17.9	4.1	3.2	1.9	11.1	18.3	36.9	29.2	7	0.64
	55.5	12.3	8.6	4.5	21.3	50.1	131.4	102.8	21	
PC1	22.8	5.6	3.3	2.8	15.2	20.7	60.8	46.1	9.9	0.88
	118.3	30.4	16.8	12.5	29.8	94.1	396.7	309.9	55.6	

The quality of the clusters is evaluated through Silhouette score [69]. Silhouette score ranges from -1 to 1 representing a worst to the best clustering respectively. We recommend dropping the dataset if Silhouette is less than 0.8. The reason is significant disjointness that can lead to reliable labeling.
The proposed framework was used to develop two clusters on each of seven datasets. So, a total of 14 clusters were generated. The initial instance and random seed was modified to attain as much high Silhouette score as possible. It implied to have more coherent and least coupled clusters. The details of the centroid for each cluster has been presented in Table IV. Out of the seven datasets, MW1 cannot achieve a satisfactory silhouette score, therefore we do notmake it a part of the labeling process. While the rest of the six projects successfully achieved a sufficient silhouette score. The outcome of this phase is a pair of clusters in each project which is least coupled and strongly coherent with its neighboring cluster.

2) Selection of experts': Expert refers to the individual having experience in software development (SD) or software quality assurance (SQA). It is desirable to have more than three experts and all the experts should share any of these two qualities while vary in the following five traits

1) Qualification
2) Years of experience
3) Domain of experience
4) Organization type
5) Followed software development model

This diversity shall provide resilience in committing common mistakes. In our selected dataset, we designed an online and in-person survey to approach experts from diverse backgrounds.

3) Computing experts' score: All the experts are not equal in not their verdicts about declaring any centroid as *FP* or *NFP*. The verdict from more experienced personnel deserves more importance. To achieve this, we derive an equation to assigneddifferent score to each participating expert. For this we have collected two traits of expertise from each respondent:

1) Years of experience in software quality assurance - E(SQA)
2) Years of experience in software development - E(SD)

Collected information against each expert is shown in Table V. The values of E(SD) and E(SQA) are scaled using Minmax that becomes $\bar{E}(SD)$ and $\bar{E}(SQA)$. The scaling is performed to void the impact of large values (see Equation 1 and 2).

$$\bar{E}(SD)_i = Minmax(E(SD)_i) \tag{1}$$

$$\bar{E}(SQA)_i = Minmax(E(SQA)_i) \tag{2}$$

After that, the values of $\bar{E}(SQA)$ is multiplied by a factor of two. The value of two is taken as the heuristic constant to assign more importance to experience in SQA than experience in SD. Finally, the overall score of the expert is computed by adding both of these values. Mathematically it can be written as:

$$Score(E_i) = \bar{E}(SD)_i + 2 \times \bar{E}(SQA)_i \tag{3}$$

In Table V, last column shows the scores assigned by experts selected.

TABLE V

DETAIL OF EXPERTS WITH THEIR CORRESPONDING SCORE

i	$E(SQA)$	$E(SD)$	$\bar{E}(SQA)$	$2 \times \bar{E}(SQA)$	$\bar{E}(SD)$	$Score$
1	2	3	0.06	0.11	0.1	0.21
2	1	3	0.03	0.06	0.1	0.16
3	2	3	0.06	0.11	0.1	0.21
4	0	8	0	0	0.27	0.27
5	0	3	0	0	0.1	0.1
6	8	2	0.23	0.46	0.07	0.52
7	20	0	0.57	1.14	0	1.14
8	20	3	0.57	1.14	0.1	1.24
9	10	0	0.29	0.57	0	0.57
10	2	12	0.06	0.11	0.4	0.51
11	2	6	0.06	0.11	0.2	0.31
12	35	30	1	2	1	3
13	0	12	0	0	0.4	0.4
14	3	15	0.09	0.17	0.5	0.67
15	0	20	0	0	0.67	0.67
16	3	8	0.09	0.17	0.27	0.44
17	0	7	0	0	0.23	0.23
18	14	20	0.4	0.8	0.67	1.47
19	8	20	0.23	0.46	0.67	1.12
20	12	5	0.34	0.69	0.17	0.85
21	8	0	0.23	0.46	0	0.46
22	10	12	0.29	0.57	0.4	0.97

4) Collecting response for preliminary labeling: The centroid constitutes the metrics with their corresponding values. Preliminary labeling is performed by providing such centroid information to the expert. The outcome of this phase is the experts' label to one centroid as *FP* and another as *NFP*, which we named as preliminary labeling.

5) Computing centroid scores: Since there must have multiple experts and can have conflict in responses the question arises, which label should we assign to a centroid? We address this issue by first assigning score to each expert in section III-B3, based upon which, we assigned score to the centroid which is labeled by the experts. The score of a centroid C_k is computed using equation 4.

$$Score(C_k) = \sum_{i=1}^{n} Score(E_i) \times L_i \tag{4}$$

Where, L_i is the label assigned by an i^{th} expert to C_k centroid, which is one in case of *FP* and zero in case of *NFP*, and n is the total number of centroids, which is 12 in our case.

6) Computing difference of opinion: We evaluated the labeling quality by computing **D**iffererence **o**f o**P**inion (DoP) using equation 5.

$$DoP = \frac{\mathbf{min}\{Score(C_1), Score(C_2)\}}{\mathbf{max}\{Score(C_1), Score(C_2)\}} \times 100 \tag{5}$$

Where, C_1 and C_2 are two centroids of a datasets. Score of the centroids were computed using equation 4. The computed DoP for in CM1, KC1, KC2, KC3, MC2, and PC1 are 0.2, 0.1, 0.12, 0.13, 0.11, 0.17, and 0.15 respectively, which is less than 20%, therefore we disregard the difference of opinion amongst experts.

7) Permanent labeling: For permanent labeling to the centroid, we use the following simple rule.

if $Score(C_1) > Score(C_2)$ **then**
  $C_1 = FP$
  $C_2 = NFP$
else
  $C_2 = FP$
  $C_1 = NFP$
end if

Using the above rule, we declared one centroid as FP, while the other as NFP, and then the label of a centroid is assigned to all data points of the corresponding cluster.

IV. COMPARATIVE ANALYSIS

EkmEx extends the base technique of Clustering with expert opinion while overcoming its associated shortcomings. We believe that the EkmEx could improve the labeling process by making the process more structured and thus reliable. For the validation and evaluation purpose, we build a ML model using Random forest implementation of Weka [1]. An experiment is performed tocompare the performanceof EkmEx and the base approach for labeling the six datasets. The obtained results are shown in Table VI. The Table VI concludes the following results

1) The results show outperformance of EkmEx in F1 score in five datasets with mild underperformance in KC3 dataset.
2) EkmEx achieves the higher Recall in all the six datasets.

TABLE VI

PRECISION, RECALL, AND F 1 SCORE COMPUTED IN EKMEX AND THE BASE TECHNIQUE

Datasets	**Approaches**	**TP**	**FN**	**TN**	**FP**	**Recall**	**Precision**	**F-1-Score**
CM1	EkmEx	40	9	412	37	⬆ 0.82	⬆ 0.52	⬆ 0.63
	Base approach	37	12	379	70	0.76	0.35	0.47
KC1	EkmEx	312	14	1544	239	⬆ 0.96	⬆ 0.57	⬆ 0.71
	Base approach	300	26	1010	773	0.92	0.28	0.43
KC2	EkmEx	79	28	400	15	⬆ 0.74	⬆ 0.84	⬆ 0.79
	Base approach	53	54	394	21	0.50	0.72	0.59
KC3	EkmEx	40	3	397	18	⬆ 0.93	⇩ 0.69	⬆ 0.79
	Base approach	37	6	402	13	0.86	0.74	0.80
MC2	EkmEx	40	12	95	14	⬆ 0.77	⬆ 0.74	⬆ 0.75
	Base approach	37	15	56	53	0.71	0.41	0.52
PC1	EkmEx	70	7	919	113	⬆ 0.91	⬆ 0.38	⬆ 0.54
	Base approach	67	10	756	276	0.87	0.20	0.32

V. Threats to Validity

The results of our experiment allow us to compare the proposed labeling framework EkmEx with the base technique. Before we could accept the result, we would have to consider possible threats to its validity.

1) We show the results against only six datasets. If we would include more datasets for labeling, the results will even be more generalizable. Hence, the general applicability of the results remains unrevealed.
2) With regard to the size of the projects, sufficient comprehensible project size is taken. The projects of a very large size or very small size were ignored.

3) With regard to the metrics, we have used only available metrics parsed by the other researcher, which heavily influence the clustering process. If we add more metrics or different metrics the cluster may get change and leads to a difference in results just reported.

VI. Conclusion and Future Work

Information about the *FP* and *NFP* instances are primary requirements in supervised SFP activity. However, acquiring the fault information about software products is not a normal practice. The SFP research community performed a valuable effort for labeling the modules as *FP* or *NFP*. Existing techniques suffer the issues of narrow coverage, dropping instances or least reliable labeling results. This gives rise to the need of proposed approach named EkmEx in labeling the unlabeled instances and thus performing supervised SFP without prior fault information. A comparative evaluation of EkmEx was performed with base technique on seven datasets by NASA. We observed that EkmEx has a mild difference in identifying faulty instances as *FP* instances w.r.t. its contemporary technique. A significant performance improvement has been observed in asserting the clean instances as *NFP*. Similarly, EkmEx achieves a higher F1 score and precision in all the three versions than its counterpart, while mild underperformance in terms of precision and F1 score in KC2 dataset.

In future, we look forward to analyse the performance of labeling via multiclass labels as fault severity, using EkmEx.

References

[1] I. 9000:2015(en), "Quality management systems fundamentals and vocabulary," *ISO*, 2015.

[2] M. 1, "Marine 1 code error," Mariner1.https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=MARIN1;.

[3] B. Kotková and M. Hromada, "Adverse event in a medical facility-blackout," *International Journal of Power Systems*, vol. 5, 2020.

[4] A. Zakari and S. P. Lee, "Simultaneous isolation of software faults for effective fault localization," in *2019 IEEE 15th International Colloquium on Signal Processing & Its Applications (CSPA)*. IEEE, 2019, pp. 16–20.

[5] H. Alsghaier and M. Akour, "Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier," *Software: Practice and Experience*, vol. 50, no. 4, pp. 407–427, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2784

[6] S. Beecham, T. Hall, D. Bowes, D. Gray, S. Counsell, and S. Black, "A systematic review of fault prediction approaches used in software engineering," *The Irish Software Engineering Research Centre: Limerick, Ireland*, 2010.

[7] R. S. Wahono, "A systematic literature review of software defect prediction: research trends, datasets, methods and frameworks," *Journal of Software Engineering*, vol. 1, no. 1, pp. 1–16, 2015.

[8] A. Al-Shaaby, H. Aljamaan, and M. Alshayeb, "Bad smell detection using machine learning techniques: A systematic literature review," *Arabian Journal for Science and Engineering*, pp. 1–29, 2020.

[9] C. Catal and B. Diri, "A systematic review of software fault prediction studies," *Expert Syst. Appl.*, vol. 36, no. 4, pp. 7346–7354, May 2009.

[10] I. Ghani, *Handbook of research on emerging advancements and technologies in software engineering*. IGI Global, 2014.

[11] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*. CRC Press, 2014.

[12] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and Software Technology*, vol. 55, no. 8, pp. 1397–1418, 2013.

[13] S. Beecham, T. Hall, D. Bowes, D. Gray, S. Counsell, and S. Black, "A systematic review of fault prediction approaches used in software engineering," Technical Report Lero-TR-2010-04, Lero, Tech. Rep., 2010.

[14] I. Gondra, "Applying machine learning to software fault-proneness prediction," *Journal of Systems and Software*, vol. 81, no. 2, pp. 186–195, 2008.

[15] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, no. C, pp. 504–518, Feb. 2015.

[16] T. Chappelly, C. Cifuentes, P. Krishnan, and S. Gevay, "Machine learning for finding bugs: An initial report," in *Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE), IEEE Workshop on*. IEEE, 2017, pp. 21–26.

[17] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, 2017.

[18] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *Institution of Engineering and Technology Software*, vol. 12, no. 3, pp. 161–175, 2018.

[19] M. H. Halstead, *Elements of Software Science (Operating and Programming Systems Series)*. New York, NY, USA: Elsevier Science Inc., 1977.

[20] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, Jul. 1976.

[21] L. C. Briand, J. Daly, V. Porter, and J. Wust, "A comprehensive empirical validation of design measures for object-oriented systems," in *Proceedings Fifth International Software Metrics Symposium. Metrics (Cat. No.98TB100262)*, Nov 1998, pp. 246–257.

[22] K. El Emam, S. Benlarbi, N. Goel, and S. Rai, *A validation of object-oriented metrics*. National Research Council Canada, Institute for Information Technology, 1999.

[23] K. El-Emam and W. Melo, "The prediction of faulty classes using object-oriented design metrics," *Journal of Systems and Software*, vol. 56, 02 2001.

[24] S. S. Rathore and S. Kumar, "A decision tree logic based recommendation system to select software fault prediction techniques," *Computing*, vol. 99, no. 3, pp. 255–285, 2017.

[25] C. Catal, "Software fault prediction: A literature review and current trends," *Expert systems with applications*, vol. 38, no. 4, pp. 4626–4636, 2011.

[26] T. Zimmermann and N. Nagappan, "Predicting defects using

network analysis on dependency graphs," in *2008 ACM/IEEE 30th International Conference on Software Engineering*, May 2008, pp. 531–540.

[27] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[28] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a fault prediction model to allow inter languagereuse," in *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*, ser. PROMISE '08. New York, NY, USA: ACM, 2008, pp. 19–24.

[29] K. Li, Z. Xiang, T. Chen, S. Wang, and K. C. Tan, "Understanding the automated parameter optimization on transfer learning for cpdp: An empirical study," *arXiv preprint arXiv:2002.03148*, 2020.

[30] Z. Xu, S. Pang, T. Zhang, X.-P. Luo, J. Liu, Y.-T. Tang, X. Yu, and L. Xue, "Cross project defect prediction via balanced distribution adaptation based transfer learning," *Journal of Computer Science and Technology*, vol. 34, no. 5, pp. 1039–1062, 2019.

[31] A. AbuHassan, M. Alshayeb, and L. Ghouti, "Software smell detection techniques: A systematic literature review," *Journal of Software: Evolution and Process*, p. e2320, 2020.

[32] T. Hall, M. Zhang, D. Bowes, and Y. Sun, "Some code smells have a significant but small effect on faults," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 4, Sep. 2014. [Online]. Available: https://doi.org/10.1145/2629648

[33] Shi Zhong, T. M. Khoshgoftaar, and N. Seliya, "Unsupervised learning for expert-based software quality estimation," in *Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings.*, March 2004, pp. 149–155.

[34] J. Nam and S. Kim, "Clami: Defect prediction on unlabeled datasets (t)," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 452–463.

[35] Y. Yang, Y. Zhou, J. Liu, Y. Zhao, H. Lu, L. Xu, B. Xu, and H. Leung, "Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 157–168.

[36] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*. MIT Press, 2001, pp. 849–856.

[37] Y. Yang, J. Yang, and H. Qian, "Defect prediction by using cluster ensembles," in *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*, March 2018, pp. 631–636.

[38] R. Bender, "Quantitative risk assessment in epidemiological studies investigating threshold effects," *Biometrical Journal: Journal of Mathematical Methods in Biosciences*, vol. 41, no. 3, pp. 305–319, 1999.

[39] C. Catal, U. Sevim, and B. Diri, "Software fault prediction of unlabeled program modules," in *Proceedings of the world congress on engineering*, vol. 1, 2009, pp. 1–3.

[40] N. Seliya and T. M. Khoshgoftaar, "Software quality analysis of unlabeled program modules with semisupervised clustering," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 37, no. 2, pp. 201–211, 2007.

[41] K. Herzig, S. Just, and A. Zeller, "Its not a bug, its a feature: How misclassification impacts bug prediction," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE 13. IEEE Press, 2013, p. 392401.

[42] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced? bias in bug-fix datasets," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC/FSE 09. New York, NY, USA: Association for Computing Machinery, 2009, p. 121130. [Online]. Available: https://doi.org/10.1145/1595696.1595716

[43] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 382–391.

[44] B. Turhan, T. Menzies, A. B. Bener, and J. 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, Oct. 2009.

[45] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *arXiv preprint arXiv:1911.02685*, 2019.

[46] S. Herbold, "Training data selection for cross-project defect prediction," in *Proceedings of the 9th International Conference on Predictive Models in Software Engineering*, ser. PROMISE 13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2499393.2499395

[47] J. Zhang, J. Wu, C. Chen, Z. Zheng, and M. R. Lyu, "Cds: A crossversion software defect prediction model with data selection," *IEEE Access*, vol. 8, pp. 110 059–110 072, 2020.

[48] S. Amasaki, "Cross-version defect prediction: use historical data, cross-project data, or both?" *Empirical Software Engineering*, pp. 1–23, 2020.

[49] A. A. S. A. B. H. M. A.-A. Y. Jararweh, "Employing deep learning methods for predicting helpful reviews," *ICICS*, no. 7-12, 2020.

[50] W. Li and R. Shatnawi, "An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution," *J. Syst. Softw.*, vol. 80, no. 7, p. 11201128, Jul. 2007. [Online]. Available: https://doi.org/10.1016/j.jss.2006.10.018

[51] S. Olbrich, D. S. Cruzes, V. Basili, and N. Zazworka, "The evolution and impact of code smells: A case study of two open source systems," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, pp. 390–400.

[52] S. M. Olbrich, D. S. Cruzes, and D. I. K. Sjberg, "Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems," in *2010 IEEE International Conference on Software Maintenance*, 2010, pp. 1–10.

[53] D. I. K. Sjoberg, A. Yamashita, B. Anda, A. Mockus, and T. Dyba, "Quantifying the effect of code smells on maintenance effort," *IEEE Trans. Softw. Eng.*, vol. 39, no. 8, p. 11441156, Aug. 2013. [Online]. Available: https://doi.org/10.1109/TSE.2012.89

[54] R. Gupta and S. K. Singh, "Using software metrics to detect temporary field code smell," in *2020 10th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, 2020, pp. 45–49.

[55] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[56] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten, "'neural-gas' network for vector quantization and its application to time-series prediction," *IEEE Transactions on Neural Networks*, vol. 4, no. 4, pp. 558–569, 1993.

[57] D. Rodriguez, R. Ruiz, J. C. Riquelme, and R. Harrison, "A study of subgroup discovery approaches for defect prediction," *Inf. Softw. Technol.*, vol. 55, no. 10, p. 18101822, Oct. 2013. [Online]. Available: https://doi.org/10.1016/j.infsof.2013.05.002

[58] G. Boetticher, T. Menzies, and T. Ostrand, "{PROMISE} repository of empirical software engineering data," *ArXiv*, 01 2007.

[59] D. M. L. M. R. Romain, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4, pp. 531–577, Aug 2012.

[60] X. Yuan, T. M. Khoshgoftaar, E. B. Allen, and K. Ganesan, "An application of fuzzy clustering to software quality prediction," in *Proceedings 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, 2000, pp. 85–90.

[61] J. Yang and H. Qian, "Defect prediction on unlabeled datasets by using unsupervised clustering," in *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Dec 2016, pp. 465–472.

[62] M. Yan, Y. Fang, D. Lo, X. Xia, and X. Zhang, "File-level defect prediction: Unsupervised vs. supervised models," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Nov 2017, pp. 344–353.

[63] R. Marinescu, "Detection strategies: metrics-based rules for detecting design flaws," in *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, Sep. 2004, pp. 350–359.

[64] C. Catal, U. Sevim, and B. Diri, "Clustering and metrics thresholds based software fault prediction of unlabeled program modules," in *2009 Sixth International Conference on Information Technology: New Generations*, April 2009, pp. 199–204.

[65] P. S. Bishnu and V. Bhattacherjee, "Software fault prediction using quad tree-based k-means clustering algorithm," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 6, pp. 1146–1150, June 2012.

[66] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, Sept 2013.

[67] L. Son, N. Pritam, M. Khari, R. Kumar, P. Phuong, and T. Pham, "Empirical study of software defect prediction: A systematic mapping," *Symmetry*, vol. 11, p. 212, 02 2019.

[68] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," *Communications of the ACM*, vol. 32, no. 12, pp. 1415–1425, Dec. 1989.

[69] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 11 1987.