An Exploratory Framework for Intelligent Labelling of Fault Datasets

Muhammad Rizwan
Department of Computer Science
GC University
Lahore, Pakistan
rizwanabuahmad@gmail.com

Zulfiqar Habib
Faculty of Information and Technology
COMSATS University
Islamabad, Pakistan
drzhabib@cui.edu.pk

Sohail Sarwar
Faculty of Information and Technology
COMSATS University
Islamabad, Pakistan
sohail.sarwar@seecs.edu.pk

Muddesar Iqbal
Department of Computer Science
London South Bank University
London, Engalnd
m.iqbal@lsbu.ac.uk

Muhammad Safyan
Department of Computer Science
GC University
Lahore, Pakistan
m.safyan@gcu.edu.pk

Dr. Dhafer Almakhles
College of Engineering
Prince Sultan University
Riyadh 11586, Saudi Arabia
dalmakhles@psu.edu.sa

***Abstract*—Software fault prediction (SFP) has become a pivotal aspect in realm of software quality. Nevertheless, discipline of software quality suffers the starvation of fault datasets. Most of the research endeavors are focused on type of dataset, its granularity, metrics usedand metrics extractors. However, sporadic attention has been exerted on developmentof fault datasets and their associated challenges. There are very few publicly available datasets limiting the possibilities of comprehensive experiments on way to improvising the quality of software. Current research targets to address the challenges pertinent to fault dataset collection and developmentif one is not available publicly. It also considers dynamic identification of available resources such as public dataset, open-source software archieves, metrics parsers and intelligent labeling techniques. A framework for datasetcollection and development process has been furnished alongwith evaluation procedure for the identified resources.**

***Index Terms*—Software bugs, Software faults, Software metrics, Metrics extractor, Threshold, Expert Opinion**

I. Introduction

SFP is the process of identifying the fault prone modules of a software system. It allows practitioners to optimally allocate the testing resources on parts that are identified as *faulty* compared to other modules of the software system. This leads to the reduction of testing time and improves its performance. SFP models are developed using software fault datasets. These SFP models can be applied to software under development/test. Hence practitioners can identify defect-prone parts of a software system. SFP research community is more active in recent years than ever before[1, 2]. Likewise,reviews, SLRs, and mapping studies have also been published quite frequently[1–20]. Such aspects are clear evidence for acceptance and indispensability of SFP process. However, software defect dataset is one of the primaryrequirements for process of SFP[8].

SFP empirical studies have been observed of inclination towards quality of data and suffer from limited generalizations. The main reasons are non-availability of data and systematic procedures of data collection. Dataset is a collection of multiple records/instances that comprises of one or more metrics and a label. The label is either a dichotomous variable(such as presence or absence of fault) or a numeric variable (such as number of faults). A wide variety of datasets has been used for SFP that have been divided into three categories based upon their availability: public, partially public, private [15]. *Public* datasets refer to the dataset wherein the metrics values and the fault data is publicly available for all the modules in a software system. In a *Partially public*, usually, the source code and fault data is available, but not the metrics values, which need to be extracted from the source code and mapped with the fault data from the repository [15]. Finally, the *Private* datasets do not provide source code as well as fault information, so the studies based on these datasets may not be repeatable. In addition to these three types, we have identified two more types: *Partially private* and *Absolute public*. In *Partially private* only source code and/or metrics values are available without fault information. However, we need to employe some existing labeling techniques for dataset labeling. This type is further split into three different types based upon the availabilityof any of metrics values and/or source code. Finally, *Absolute public* is the public dataset having source code availability also. Table I shows five types of dataset. The SLR from years 1991 to 2013 concludes that 23% of the papers use a private dataset, while 87% of papers use public dataset [12]. This implies that researchers are more inclined towards public datasets. The reason is the least availability of fault information, in other words, lack of labeled dataset. Catal and Diri [20] conduct a systematic literature review (SLR) from the years 1990 to 2007. They conclude that 69% of papers have private datasets published before 2005, which becomes 48% after 2005. SLR conducted by Radjenvic *et al.* [15] covers the studies published between 1991 and 2011 declares that 62% of the studies used private, 22%

partially public and only 22 public datasets. Ronald reports that from 1996 to 2012, 57% of the studies used publicly available datasets. Likewise, [14] concludes that 64.89% of the research studies published from Jan 2000 to Dec 2013 used public datasets and 35.21% of the research studies used private datasets.

TABLE I
DATASETS TYPES W.R.T. AVAILABILITY OF METRICS VALUES, FAULT INFORMATION, AND SOURCE CODE

Type of dataset		Metrics' values	Fault information	Source code
Absolute private		✗	✗	✗
Private		✗	✓	✗
Partially Private	level-1	✓	✗	✗
	level-2	✗	✗	✓
	level-3	✓	✗	✓
Partially public		✗	✓	✓
Public		✓	✓	✗
Absolute public		✓	✓	✓

This collectively implies that researchers are more inclined towards public datasets. The reason is the least availability of sources for developing partially public/private datasets. The sources include (but not limited to) fault information, software source code, metrics extractor, etc. The research community performs sporadic attention to the problem. Mostly, the studies discuss the type of dataset, its granularity [2, 3, 5, 10], metrics used and sometimes metrics extractor[4, 9, 10, 21]. However, there is a lack of comprehensive study that addresses the issues specific to datasets i.e. identification public dataset and how to develop new dataset if one is not publically available.
The objective of current research is to provide the information about basic sources of fault datasets and then providing a framework for developing SFP datasets, in case of non-availability of sources to perform the SFP task.
Rest the document is structured as follows: Section II briefly discusses the problems related to datasets. Section III describes the methodology of current work. Section IV identifies the public dataset. In the absence of a public dataset alternate option is to use software archives, which is discussed in Section V and parsing metrics in Section VI. Since SFP requires the labeled dataset. This requirement can be met by either relating the fault information achieves or synthetic labeling techniques, which are discussed in Section VII.

II. LITERATURE REVIEW

SFP research community is more active in the recently years than ever before[1, 2]. Likewise,reviews, SLRs, and mapping studies are also frequently being published. Since our focus is on the dataset related aspect sporadic attention has been given to it. This section briefly discussed such studies in reverse chronological order.
Shabby *et al.* [3] focused on the code smell related studies and the dataset used. Amongst the seventeen studies, fourteen studies used open-source systems, while three studies used industrial systems. The article identifies the metrics and size of datasets used in each study.
Canedo *et al.* [4] performed SLR on the studies published from 2007 to 2018. The authors identify metrics extractor parsing used in the selected studies.
Hoang *et al.* [5] identified 98 studies published between 1995 to 2018 while focusing on the preprocessing/modeling technique, kind of datasets, and performance metric used in the selected studies. Regarding datasets, only types of the dataset being used by the SFP community are discussed which include public and private datasets. They further classified the used datasets into seven classes; Apachi, Promise, Mozilla, Eclipse, NASA, Student developed, and others.
Karu and Singh [6] identified the datasets used for the investigating of refactoring activities on software quality. The authors also provided an insight into the tools available for code smell. The article reported that 26 primary studies provide sufficient detail of the datasets used in 142 primary studies with 294 distinct datasets. Since, 15 datasets are used by more than three studies. They provide three types of information of the 15 datasets; datasets name, programming language, dataset type, and studies used them. The article included only three studies related to fault prediction.
Sandeep and Santosh[7] discussed various dimensions of SFP. However, the study has a very shallow focus on the types of datasets, metrics extractors, and public source code and bug repositories.
Li *et al.*[8] listed ten datasets repositories used in the studies published from January 2014 to April 2017, while providing few properties of each repository like number of projects, metrics' coverage, granularity, etc.
Mayra Nilsson [9] conducted the most comprehensive known study on available metrics extractors. The author identified and evaluated 130 commercial and non-commercial metrics extraction for multiple languages such as .NET, Ada, C, C++, Java, JAvaScript, Perl, PHP, Phyton, and Ruby, etc. The author informed about the metrics covered by the identified extractors. While drawing six evaluation criteria the author shortlisted eight metrics extractors; QAC, Understand, CP-PDepend, SourceMeter, SonarQube, Eclipse Metrics Plugin, CodeSonar, and SourceMonitor.
A study conducted by [10] aimed to illustrate the taxonomy of the methodologies used in SFP. The authors listed the four datasets frequently used by the SFP community along with the list of available metrics extractor.
S. S. Rathore [11] provided an overview of the public, partial public, private datasets used since 1993 to 2017.
Hosseini *et al.* [2] provided a comprehensive list of public and private datasets used in the 46 primary studies published in the domain of cross-product defect prediction until 2015.
R. Malhotra [12] drawn five classes of datasets used in the studies published from January 1991 to October 2013. The authors briefly described the datasets and their corresponding studies.
R. S. Wahono [14] performs an SLR on the studies published from Jan 2000 to Dec 2013. The objective of the studies to analyze the datasets, methods, and frameworks used by the

SFP research community.

Mausa *et al.* [13] performed a study on approaches and tools used in the development of SFP datasets. The authors provided a detail data collection procedure while identifying and analyzing a comprehensive list of 35 extractors for software product metrics.

Ronald *et al.* [?] identifies 95 primary studies in their SLR. The authors listed all the dataset used in the primary studies along with the programming language of corresponding datasets.

Radjenović *et al.* [15] selected 106 primary studies published from 1991 and 2011. The authors reported that 62 of the studies used private, 22 partially public and only 22 public datasets. They identified the programming languages and software development life cycle used to implement software from which data sets are extracted.

Bassey and Obeten [16] performed SLR and included 29 studies. They identified metrics extractors for building private or partial public datasets. Moreover, they listed the public datasets used by the studies. They reported that industrial software systems are used by 79% of the studies while non-industrial (student) software systems are taken for the collection of datasets in 21% of the applications. Moreover, they concluded that programming language of the software systems used in the studies are C++, Java with 54% and 43% respectively.

R. S. Wahono [17] plotted the percentage usage of private and public datasets across the studies published between 2000 and 2013. Moreover, the authors provided information regarding trends of usage of private and public datasets over time.

Hall *et al.* [18] discussed the Quality of dataset and fault severity carrier datasets used by 208 studies published from January 2000 to December 2010.

C. Catal [1] investigated 90 software fault prediction papers published between year 1990 and 2009. The authors briefly described the datasets with their types and metrics used in the primary studies.

Kayarvizhy and Kanmani [?] design an automated object oriented metrics extractor having generic framework. The extractor converts the source code into XML format, from which metrics can be computed. The authors took two extractors CKJM and JMT to compare the metric values and compare results with their extractor.

Catal and Diri [19] investigated the performance of machine learning algorithms on large and/or public datasets when different metrics metrics selection techniques are employed.

Catal and Diri [20] identified the kind and distribution of dataset which are mostly used for fault prediction in the studies published before and after 2005.

Mostly, the studies discuss the type of dataset, its granularity, metrics used and sometimes metrics extractors are discussed. However, there is a missing comprehensive study that addresses the issues specific to datasets i.e. available public dataset and how to develop new dataset if one is not publically available. Our study differs from the existing studies in both the aim and scope of the selected studies. The objective of this article is to provide the available resources and then providing tools, techniques, and methodologies to address if the required resource is not available in building a dataset to perform the SFP task.

III. Methodology

We put an effort to develop a systematic solution of the problem, which is shown in Figure 1. After that, we disclose and resolve every trap in the process that can benefits the body of knowledge.

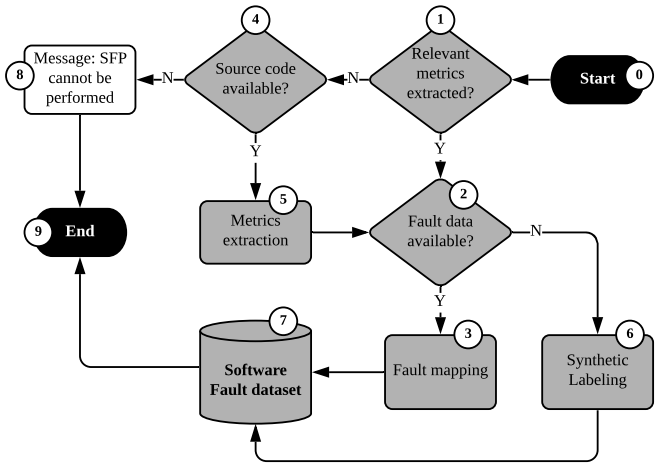

Fig. 1. Roodmap to meet the requirement of fault dataset

TABLE II
Dataset to Path Mapping

Paths (Best to worst)	Absolute private	Private	Partially private (Level 1)	Partially private (Level 2)	Partially private (Level 3)	Partially public	Public	Absolute public
012379							✓	✓
01452379						✓		✓
012679			✓		✓		✓	✓
01452679				✓	✓	✓		✓
01489	✓	✓						

Keeping in view the roadmap to the problems' solutions, we first identify the sources of public datasets, which is expected to have metrics' information along with fault mapping. However, in the absence of such a dataset, the procedure is shown in Figure 1 guides to develop dataset. That comprises the recognization of the open-source software achieves for the input to metrics extractors. Once the metrics information is collection labeling is done either by analyzing fault information (if it is available) or applying synthetic labeling techniques (if fault information is unavailable).

We can be lucky to have multiple sources to build fault data. In such scenario, we can follow multiple paths to the graph shown in Figure 1. Like, if we have absolute public dataset, we can follow paths {012379}, {01452379}, {012679}, and {01452679}. Table II shows the possible paths that can be followed in the presence of corresponding specific datasets. However, in such case we should follow the best path. The rows in the Table II is sorted from best to worst path.

Therefore, one should follow the best solution in the presence of multiple possibilities.

We believe the procedure is applicable in industrial research as well as in academics. We review the requirements, limitations, and available resources to SFP community related to datasets collection/development. More specifically followings are our research objective:

1) Identifying the public dataset to perform SFP activities.
2) What are tools for extracting metrics from the software source code?
3) Identifying the open-source software archives with/without fault information?
4) Identifying the techniques to associate fault information with the metrics data
5) Identifying the fault labeling methodologies/techniques proposed/used by the SFP community.

Keeping in view the above objectives we draw the following inclusion/exclusion criteria:

1) The datasets that comprise only the static product code metrics are included. However, the datasets that comprise process metrics are excluded.
2) The tools that extract static code metrics are included. Therefore metrics extractor that requires other software artifacts (like use case diagrams etc.) are excluded, like SDMetrics[22]. This also excludes the tools which are used to extract dynamic/semantic metrics or used to visualize metrics from the source.

Since our primary focus is on the dataset exclusively, Section IV identifies the public dataset. In the absence of a public dataset alternate option is to use software archives (Section V) and parsing metrics[1] using metrics extractor (Section VI). Since SFP requires the labeled dataset. This requirement can be met by either relating the fault information achieves or synthetic labeling techniques, which are discussed in Section VII.

IV. Public datasets repositories

Public datasets refer to the dataset wherein the metrics values and the fault data is publicly available for all the modules in a software system. The SLR from years 1991 to 2013 concludes that 23% of the papers use a private dataset, while 87% of papers use public dataset [12]. This implies that researchers are more inclined towards public datasets. The reason is the least availability of fault information, in other words, lack of labeled dataset. Catal and Diri [20] conduct a systematic literature review (SLR) from the years 1990 to 2007. They conclude that 69% of papers have private datasets published before 2005, which becomes 48% after 2005. SLR conducted by Radjenvic *et al.* [15] covers the studies published between 1991 and 2011 declares that 62% of the studies used private, 22% partially public and only 22 public datasets. Ronald [**?**] reports that from 1996 to 2012, 57% of the studies used publicly available datasets. Likewise, [14] concludes that 64.89% of the research studies published from Jan 2000 to Dec 2013 used public datasets and 35.21% of the research studies used private datasets. This collectively implies that researchers are more inclined towards public datasets. This section is dedicted to identify the availabe publicized datasets to perform SFP activity.

1) NASA [23] dataset repository consists of 13 software projects in PROMISE version and 14 software projects in MDP version, with considerable difference in the number of features set also [24]. Besides having the quality problems [24], it is the most used dataset by the SFP community [25].
2) SOFTLAB [26] dataset contains five projects, i.e., AR1, AR2, AR3, AR4, and AR5, which are embedded controller software for white goods. All these projects are written in C.
3) AEEEM [27] is collected by D'Ambros *et al.* sometime named as Bug prediction dataset [28]. It contains class level datasets of five software systems; *Eclipse JDT Core, Eclipse PDE UI, Equinox Framework, Lucene and Mylyn*. These datasets have 61 metrics, which include code metrics, process metrics, and churn metrics.
4) ReLink [29] is a file-level dataset of three projects; Apache, Safe, and ZXing having 194, 56, and 399 instances, respectively. Each dataset contains 26 features, including complexity and count metrics.
5) Jureczko and Madeyski[30] collected 92 versions of 38 different software development projects. These projects are open source, proprietary and academic software projects. Each of these datasets contains 20 metrics, including McCabe's cyclomatic metrics, CK metrics, and other OO metrics. Few of these datasets are Tomcat, Ant, Camel, Ckjm, Forrest, Ivy, JEdit, Log4j, Lucene, PBeans, Poi, Prop, Synapse, velocity, Xalan, Xerces, etc.
6) ECLIPSE [31] contains file and package level datasets of three versions of Eclipse, i.e., 2.0, 2.1, and 3.0 released on 27^{th} June 2002, 27^{th} March 2003, and 25^{th} June 2004 respectively. Zimmermann *et al.* computed code metrics on the file level and 40 metrics on the package level to detect pre- and post-release defects. However, the last extension/modification was made on March 2010.
7) AEV data set was collected by Altinger et al. [32]. It is a novel industry dataset obtained from three different automotive embedded software projects developed by Audi Electronics Venture GmbH. Each project has a total of 29 software metrics.
8) Columba [28]. Columba is an email client hosted at sourceforge.net. The project migrated to SVN in July 2006 without migrating the CVS history. Sunghun *et al.* learned from the sourceforge.net support that keeps old versioning data, even when project switch to a new system.
9) Scarab is an issue tracker that is hosted at tigris.org whose CVS repository is not publicly available anymore. As for ArgoUML, Jack Repenning from tigris.org sent us the archive.

[1]This is sometimes referred to as mining software repositories(MSR)

10) Bug catcher dataset [33]. It is based upon three open-source systems. Eclipse, ArgoUML, and Apache Commons
11) GitHub Bug Dataset [**?**]. is a collection of 15 Java systems from GitHub and constructed a bug dataset at class and file level. These systems are Android Universal Image Loader, Antlr 4, Broadleaf Commerce, Ceylon IDE Eclipse Plugin, Elasticsearch, Hazelcast, JUnit, MapDB, mcMMO, MCT, Neo4J, Netty, OrientDB, Oryx, and Titan.

In addition to the above public repositories, Chakkrit Tantithamthavorn collected all the public bug datasets and make it available [34].

V. Open source software arhieves

There are two frequently used repositories Eclipse [31] and Mozilla used by numerous studies [40, 41, 65, 66, 77, 85]. Besides these Mockus [94] is a suite that contains the defect data for about 235K projects hosted on SourceForge and GoogleCode [2]. Few of such repositories are: Mozilla [95] , Apache, Eclipse, KDE, PostgreSQL [28]. Besides this

1) Savannah
2) Google Code
3) Sourceforge
4) Git
5) FreshMeat
6) Free Software Foundation (FSF)
7) Ruby- Forge
8) ObjectWeb[2]

This type of repositories mainly contained the source code for different software artifacts. Open-source software systems provide a facility to the researchers for collecting the datasets and building their own software fault dataset. However, there is one important requirement i.e. software system may be developed using a bug tracking system and have the provision of storing software development information. In presence of fault archieves, researchers need to mine these repositories for collecting the software fault dataset. The key problem with collecting datasets from the open-source systems is that it consumes an ample amount of resources. So, it is very difficult to ensure the accuracy of the dataset. Having these practical problems in obtaining software fault data, a lot of effort is made to propose alternative approaches for labeling the fault data.

VI. Metrics extractors

Once the open source archieves are identified, next step is to extract the required metrics. The availability of metrics extractor significantly eases the task of code analysis. This section aims to emumerates metrics extractors out of static code.

Table III depicts various such extractors that are available for the the research community. Availability can have three possible values; Properietry (P) or Free, whereas N/A is marked where authentic status of availability can not be identified. Some of the extractors support multiple languages [78, 90], however, we mentioned only the well known programming lanagues. Same is done in case of metrics support [35, 38, 62, 63, 70, 82, 86]. Most of the extractors compute the metrics related to static aspect of object oriented paradigm. There is a good number of free extractors supporting multiple languages. Last column shows the studies which used the corresponding extractor. However, metrics extractors have three main limitations:

1) Properietry extractors have lack of extensibility, hence may not be used to integrate with existing frameworks/extractors.
2) Extractors are usually language specific and may not be able to work on all the programming languages.
3) Extractors are metrics specific and may not parse new metrics. Hence they are not easy to adapt to other metrics.
4) Extractors have ambiguous interpertation of some metrics. Hence, more than one variant of the same metric exists which is reported in [9].

The table depicts that there are multiple extractors can be used to extract the same metrics, however, additional criteria can be applied to further find the most useful extractor. These criteria can be availability of IDE, reporting mechanism, etc. Most of the properietry extractors provide limited funcationalities like reports cannot be saved or printed or exported. Keeping in view the constraints may narrow the selection, as is done in [9].

VII. Labeling methologies

A. Intelligent Labeling through Fault Mapping

Software source code repositories having fault data is used to build the datasets by parsing metrics information from the source code and then associating the metrics' values with the fault data. The task of the association is quite challenging. Kim Herzig and Sascha [96] analyzed seven thousand issue reports in five open-source projects. They found that more than 40% of a fixed set of issue reports are inaccurately classified, with 33.8% of all "bug reports" bug database misclassified the reports. Due to this reason, 39% of files are classified as faulty actually never had a bug. Likewise, Bird *et al.* reported that out of 24 thousand fixed bugs, only 10 thousand could be linked to an entry in the bug database [97]. Aranda and Venolia [98] manually inspected ten bug reports in Microsoft and interviewed developers related to the reports. They found lots of important information missing in bug reports. Some of the reasons (but not limited to) for such ambiguity and overly impreciseness are:

1) Some developer has a habit of committing all pending local changes before the weekend in one single transaction. If any changes are classified as a fix, then all contained modules will be marked as having had a defect in the past even if only one of them was actually defective. [99]

[2]While there is clear benefit of open source code, yet there is a need to consider the concerns of the parties that provide the data to pertain the ethics associated.

TABLE III
BRIEF SUMMARY OF THE METRICS EXTRACTORS

Metrics extractor	Availability	Language	Metrics coverage	Used by
AEA tool[35]	N/A	Java	NOC, DIT	-
Amadeus[36]	N/A	C++	35 code metrics	[37]
AMT[?]	N/A	C# and Java	CBO, RFC, MPC, DAC	
Borland Together[38]	Free	Java	CK suite, CC, etc	[39–42]
Brooks and Buell's tool [43]	N/A	C++	CK suite	
CCCC[44]	Free	C++ and Java	LoC, McCabe, CK suites, Fan-in, and Fan-out	-
CCMETRICS[45]	Free	Java	Coupling and Cohesion metrics	
ckjm[46]	Free	Java	WMC, DIT, NOC, CBO, RFC, LCOM, Ca, NPM	[47, 48]
Columbus[49]	P	C++	CK suite, LOC	[50, 51]
Concerto/ AUDIT[52]	N/A	C++	CK and Briand Suite, Cohesion, inheritance metrics.	[53]
Dependency Viewer[54]	Free	Java	Ca and Ce	
Fraunhofer IESE[55]	P	C++	CK and Briand Suite, Cohision metrics	[56]
FrontEndART[57]	P	C, C++, C#, Java, and Python	Cohesion, Complexity, Coupling , Inheritance, and Size metrics	-
Ghamdis Tool[35]	N/A	Java	23 metrics including, CK suite, compelxity metrics, etc.	-
JArchitect[58]	P	Java	82 code metrics	-
Java static analysis tool	N/A	Java	CK suite	[59]
JCAT[60]	N/A	Java	PCC, ECC, GCCC, and ICCC	
JCTIViz[61]	N/A	Java	CBO and CTI	
JDepend[62]	Free	Java	Martin metrics, No. of classes, Package dependency, etc.	-
JHawk[63]	Free	Java	Halstead, McCC, LoC,LCOM, MPC, Fan-out, Fan-in, CK, and Martin suite etc.	[64–66]
JMCT[67]	Free	Java	CBO, RFC	
JMetric[68]	Free	Java	Martins suite, and few coupling metrics.	[69]
JMT[70]	Free	Java	CK suite, PIM, NMI, NAI, NMO, NOP, LOC, MIF, AIF, COF, etc.	-
Krakatau[71]	N/A	C, C++, and Java	About 70 metrics, including Halstead, complexity, OO metrics	[39]
M-System[55]	P	C++	28 coupling measures, 10 cohesion measures, and 11 inheritance measures.	[72–75]
MaX[76]	P	C and C++	Function calls, imports, exports, RPC, COM, and Registry access	[77]
McCabe IQ[78]	P	C, C++, C#, Java, VB, etc.	McCabe metrics	
OOMetDaGa Environment [79]	N/A	C++, Java, Smalltalk	CK suite	
OOMeter[80]	N/A	C# and Java	CK suite, Cohision metrics, and LOC	
Rational Rose[81]	P	XML	CK suite	[75]
Sonargraph-Explorer[82]	P	C, C++, C#, Java, and Python	Fan-in, Fan-out, Instability, Cyclicity, Component Dependencies, etc.	[83]
srcML[84]	P	C, C++ and Java	CK suite and LOC	[85]
JCMT[86]	P	Java	CK suite, Martin suite, CC, LOC, etc.	-
TAC++ [87, 88]	P	C++	15 metrics	-
Ucinet[89]	P	C and C++	Dependency graph	[77]
Understand[90]	P	C++, Java, FORTRAN, etc.	Project Metrics, File Metrics OO Metrics, Program Unit Metrics, etc.	[91]
WebMetrics[92]	P	C++	CBO, RFC, Fan-in, and Fan-out	[50, 93]

2) Bug archives do not explicitly state which version the patch file was applied.
3) Sometime bugs associated with classes that are generated at run-time. This would lead to the erroneous association of the bugs to static classes.
4) Mostly, association depends upon the time-based intervals, which is susceptible to the erroneous mapping of data to bugs.
5) Defect collection practices, which are based on 'bug' like, keywords or bug report links in changelogs do not specify the module being addressed, therefore include noises [100].
6) Given a set of linked commits, there is no way to know if commit feature bias exists, lacking access to the full set of bug-fix commits [97]

Due to any of the above reasons all the bug-fixing commits cannot be identified without extensive and costly effort [97]. As for as the automation is concerned, there is no completely accurate way in which to collect fault data from open-source projects [101]
Nevertheless, few authors put effort to standardize the procedure of bug information extraction from the open-source repositories and association with the data [13, 31]. The defect information of many software systems is managed in Bugzilla [102]. Bugzilla is a Web-based general-purpose bug tracker that allows developers to keep track of both defects that cause loss of functionality and requirements requests.

B. *Synthetic fault labeling*

In SFP studies label can be a dichotomous variable (Faulty or Clean) or it can be a number of faults. This section aims

to elaborate different methods to accomplish the labeling task. These methods can broadly be catagories into five types;

1) Cross product defect prediction [103]
2) Bad Smell based defect labeling [33]
3) Clustering and expert opinion [104, 105]
4) Heuristic based approaches [106–109]
5) Threshold based approaches [110, 111]

1) Summary of the Syntheitc labeling techniques: These studies conducted to address the specific techniques are discussed below:

(a) **Zimmermann [103]** is a well-known approach used by many researchers [103, 112–119]. However, varying distributions of datasets across the projects would not make it useful.
(b) Bug catchers [33] operate with bad smells (solely), and found that coding rule violations have a small but significant effect on the occurrence of faults at file level. Bug catchers used Bugzilla and Jira as sources of information.
(c) **Zhong *et al.* [104]** used K-means and Neural-Gas clustering methods to cluster modules, and then 15 years experienced engineer, labeled each cluster as fault-prone or not fault-prone by examination.
(d) **Seliya *et al.* [105]** perform clustering through k means and label the centroids using software engineers.
(e) **Nam and Kim [106]** propose CLA/CLAMI. The technique neither requires prior fault data nor is influenced by the problem of data distribution. However, the technique drops a significant number of instances and metrics, thus makes it nearly useless.
(f) **Yang *et al.* [107]** label their using certain beliefs supported by some other researchers. Like, highly distributed change is more likely to be a defect inducing change; likewise, larger change is expected to have a higher likelihood of being a defect-inducing change.
(g) **Andrew *et al.* [108]** 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. They proved that the connection between buggy and clean is smaller than the connection between buggy instances and the connection between clean instances.
(h) **Yan *et al.* [120]** 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. However, labeling is done based on the experiment of [107].
(i) The most recent work is done by **Yang *et. al* [109]**. The authors propose cluster ensembles and labeling approach (CEL) and apply 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 ACL [121].
(j) **B. Ralf [110]** take threshold through experience [122] and hints from literature, Tuning machine and Analysis of Multiple Versions.
(k) **Catal *et al.* [111]** clusters using X-means. Compute mean vector of each cluster which is then compared with the thresholds vector. The complete cluster is predicted as fault-prone if at least one metric of the mean vector is higher than the threshold value of that metric. X-means need kmin and kmax value. It uses a posterior probability to select the right number of clusters. {LOC, CC, UOp, UOpnd, TOp, and TOpnd} was chosen as {65, 10, 25, 40, 125, and 70} . Values are taken from Integrated software metrics (ISM). Currently, the website is not working.

TABLE IV
EVALUATION OF SYNTHETIC LABELING TECHNIQUES

Class	Study	Automatability (C/P/N)	Coverage (C/P)	Data loss (Y/N)	Difficulty (S/C)
CPDP	Zimmermann [103]	C	C	N	C
Bad Smell	Hall *et al.* [33]	C	C	N	S
Clustering and Expert Opinion	Zhong *et al.* [104]	N	C	N	C
	Seliya *et al.* [105]	N	C	N	C
Heuristic	Nam and Kim [106]	C	C	Y	S
	Yang *et al.* [107]	C	P	N	C
	Andrew *et al.* [108]	C	P	N	C
	Yan *et al.* [120]	C	P	N	C
	Yang et. al [109]	C	C	N	C
Threshold	Bender Ralf [110]	N	P	N	C
	Catal *et al.* [111]	P	P	N	C

2) Evaluation of the synthetic labeling techniques: The labeling techniques discussed in last section vary in scope and usefulness. In this section, we try to evaluate the discussed synthetic labeling techniques through six parameters. These parameters are:

1) **Automatability** implies the ability to be automate the technique. It can be complete (C), partial (P) or none(N). *Complete* automatability implies no human intervention at all, while minor intervention is pronounced as *partial* automatiability.
2) **Coverage:** Since SFP task heavily dependent on the metrics. However a labeling technique may or may not cover all the metrics. This performance measure aims to address coverage of the labeling technique. It can have two values Complete (C) coverage and partial (P) coverage. Complete coverage shows that the technique can be applied to any type of metric, while partial coverage implies that the technique can only be applied on some metrics.
3) **Data loss:**. Labeling technique is used to label data keeping in view the metrics' values provided. However, modifying the data (loss/alter) is beyond the expectation of a labeling technique. This performance measure capture the data modifying (trimming, etc.) quality of a technique. The performance measure has a dichotomous value Yes or No. The *Y* implies that the tequnique requires the data to be truncate during the process, whereas *N* shows that there is no data loss during the applicaiton of the technique.
4) **Difficulty** refers to the degree of effort requires to

implement a technique. Techniques are pronounced as complex(C) if it requires multiple techniques for implementation, otherwise, the technique is simple(S)

Table IV shows the comparison of the labeling techniques w.r.t. the performance measures. The user can adopt the technique that best suites the requirement.

VIII. Conclusion and Futurework

The usefulness of SFP is accpeted by both academia and research industry. Dataset is a key requiremnt of any SFP activity. Static-code based dataset are the most used type of dataset in the discipline. Since, few publicly available dataset and the least availability of fault information limit the possibilities of experiments, a framework to solve the dataset problem has been proposed for SFP. We put an effort to identify the available resources i.e. publicly available dataset, open-source software achieves, metrics extractors, and labeling tools/techniques. The study also provides an evaluation procedure for the identified resources.

In future, a comprehensive study is required to address the collection/development of non-static code dataset, open-source arhieves and metrics extractors. It is found that multiple extractors can be used to extract the same metrics. However, additional criteria can be applied to further find the most useful tools.

References

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

[2] S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 2, pp. 111–147, 2019.

[3] 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.

[4] E. Dias Canedo, K. Valença, and G. A. Santos, "An analysis of measurement and metrics tools: A systematic literature review," in *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 01 2019.

[5] L. H. Son, N. Pritam, M. Khari, R. Kumar, P. T. M. Phuong, P. H. Thong *et al.*, "Empirical study of software defect prediction: a systematic mapping," *Symmetry*, vol. 11, no. 2, p. 212, 2019.

[6] S. Kaur and P. Singh, "How does object-oriented code refactoring influence software quality? research landscape and challenges," *J. Syst. Softw.*, vol. 157, 2019.

[7] K. Sandeep and S. R. Santosh, *Software Fault Prediction, A Road Map*. Singapore: Springer Singapore, 2018.

[8] Z. Li, X. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, vol. 12, no. 3, pp. 161–175, 2018.

[9] M. Nilsson, "A comparative case study on tools for internal software quality measures," 2019.

[10] A. Singh, R. Bhatia, and A. Singhrova, "Taxonomy of machine learning algorithms in software fault prediction using object oriented metrics," *Procedia computer science*, vol. 132, pp. 993–1001, 2018.

[11] S. Rathore, Santosh S.and Kumar, "A study on software fault prediction techniques," *Artificial Intelligence Review*, vol. 51, no. 2, pp. 255–327, Feb 2019.

[12] 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.

[13] G. Maua, T. G. Grbac, and B. D. Bai, "Data collection for software defect prediction - an exploratory case study of open source software projects," in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2015, pp. 463–469.

[14] 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.

[15] 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.

[16] B. Isong and E. Obeten, "A systematic review of the empirical validation of object-oriented metrics towards fault-proneness prediction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 23, no. 10, pp. 1513–1540, 2013.

[17] 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.

[18] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, Nov 2012.

[19] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Information Sciences*, vol. 179, pp. 1040–1058, 03 2009.

[20] ——, "A systematic review of software fault prediction studies," *Expert Syst. Appl.*, vol. 36, no. 4, pp. 7346–7354, May 2009.

[21] E. Fregnan, T. Baum, F. Palomba, and A. Bacchelli, "A survey on software coupling relations and tools," *Information and Software Technology*, vol. 107, 11 2018.

[22] SDMetrics, "Sdmetrics," Mar 2020. [Online]. Available: www.sdmetrics.com

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

[24] 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.

[25] 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.

[26] 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. [Online]. Available: https://doi.org/10.1007/s10664-008-9103-7

[27] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Proceedings of MSR 2010 (7th IEEE Working Conference on Mining Software Repositories)*. IEEE CS Press, 2010, pp. 31 – 41.

[28] S. Kim, "Columba, eclipse jdt.core and scarab," Sep. 2015. [Online]. Available: https://doi.org/10.5281/zenodo.268448

[29] R. wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: Recovering links between bugs and changes," 09 2011, pp. 15–25.

[30] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ser. PROMISE '10. New York, NY, USA: ACM, 2010, pp. 9:1–9:10.

[31] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, ser. PROMISE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 9–. [Online]. Available: http://dx.doi.org/10.1109/PROMISE.2007.10

[32] H. Altinger, S. Siegl, Y. Dajsuren, and F. Wotawa, "A novel industry grade dataset for fault prediction based on model-driven developed automotive embedded software," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 494–497.

[33] 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

[34] "Defect data," https://github.com/klainfo/DefectData, accessed: 2020-04-05.

[35] J. AlGhamdi, M. Elish, and M. Ahmed, "A tool for measuring inheritance coupling in object-oriented systems," *information SCiences*, vol. 140, no. 3-4, pp. 217–227, 2002.

[36] A. S. Research, *Getting Started With Amadeus*. Amadeus Measurement System, 1994.

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

[38] R. C. Gronback, "Software remodeling : Improving design and implementation quality using audits , metrics and refactoring in borland," 2003.

[39] R. Shatnawi, "A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems," *IEEE Transactions on Software Engineering*, vol. 36, no. 2, pp. 216–225, March 2010.

[40] R. Shatnawi and W. Li, "The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process," *Journal of Systems and Software*, vol. 81, no. 11, pp. 1868–1882, Nov. 2008.

[41] R. Shatnawi, W. Li, and H. Zhang, "Predicting error probability in the eclipse project," in *Software Engineering Research and Practice*, 01 2006, pp. 422–428.

[42] F. B. Abreu and R. Carapuça, "Object-oriented software engineering: Measuring and controlling the development process," in *Proceedings of the 4th international conference on software quality*, vol. 186, 1994, pp. 1–8.

[43] C. L. Brooks and C. G. Buell, "A tool for automatically gathering object-oriented metrics," in *Proceedings of National Aerospace and Electronics Conference (NAECON'94)*, 1994, pp. 835–838 vol.2.

[44] tim littlefair, "C and c++ code counter," Mar 2020. [Online]. Available: https://sourceforge.net/projects/cccc/

[45] S. Husein and A. Oxley, "A coupling and cohesion metrics suite for object-oriented software," in *2009 International Conference on Computer Technology and Development*, vol. 1, 2009, pp. 421–425.

[46] D. Spinellis, "Tool writing: a forgotten art? (software tools)," *IEEE Software*, vol. 22, no. 4, pp. 9–11, 2005.

[47] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*, pp. 69–81, 2010.

[48] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, vol. 59, no. C, pp. 170–190, Mar. 2015.

[49] R. Ferenc and A. Beszedes, "Data exchange with the columbus schema for c++," in *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, 2002, pp. 59–66.

[50] A. Janes, M. Scotto, W. Pedrycz, B. Russo, M. Stefanovic, and G. Succi, "Identification of defect-prone classes in telecommunication software systems using design metrics," *Information Sciences*, vol. 176, no. 24, pp. 3711–3734, Dec. 2006.

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

[52] FAST, "Programmers manual," Mar 1997.

[53] L. Briand, J. Wst, and H. Lounis, "Replicated case studies for investigating quality factors in object-oriented designs," *Empirical Software Engineering*, vol. 6, pp. 11–58, 03 2001.

[54] M. Wilhelm and S. Diehl, "Dependency viewer - a tool for visualizing package design quality metrics," in *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2005, pp. 1–2.

[55] P. T. Devanbu, "Genoa: A customizable language- and front-end independent code analyzer," in *Proceedings of the 14th International Conference on Software Engineering*, ser. ICSE 92. New York, NY, USA: Association for Computing Machinery, 1992, p. 307317. [Online]. Available: https://doi.org/10.1145/143062.143148

[56] 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.

[57] F. S. Ltd., "Front end art," Mar 2020. [Online]. Available: http://www.frontendart.com

[58] C. Gears, "Jarchitect," Mar 2020. [Online]. Available: https://www.jarchitect.com/

[59] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, June 1994.

[60] J. Offutt, A. Abdurazik, and S. R. Schach, "Quantitatively measuring object-oriented couplings," *Software Quality Journal*, vol. 16, no. 4, p. 489512, Dec. 2008. [Online]. Available: https://doi.org/10.1007/s11219-008-9051-x

[61] P. Rosner and S. Viswanathan, "Visualization of coupling and programming to interface for object-oriented systems," in *2008 12th International Conference Information Visualisation*, 2008, pp. 575–581.

[62] jdepend, "jdepend," Mar 2020. [Online]. Available: https://github.com/clarkware/jdepend

[63] virtualmachinery.com, "Jhawk," Mar 2020. [Online]. Available: http://www.virtualmachinery.com/jhawkprod.htm

[64] M. O. Elish, A. H. Al-Yafei, and M. Al-Mulhem, "Empirical comparison of three metrics suites for fault prediction in packages of object-oriented systems: A case study of eclipse," *Advances in Engineering Software*, vol. 42, no. 10, pp. 852–859, oct 2011.

[65] K. Johari and A. Kaur, "Validation of object oriented metrics using open source software system: An empirical study," *ACM Sigsoft Software Engineering Notes*, vol. 37, pp. 1–4, 01 2012.

[66] D. Kumari and K. Rajnish, "Investigating the effect of object-oriented metrics on fault proneness using empirical analysis," *International Journal of Software Engineering and its Applications*, vol. 9, pp. 171–188, 01 2015.

[67] V. S. Bidve and P. Sarasu, "Tool for measuring coupling in object-oriented java software," *International Journal of Engineering and Technology*, vol. 8, no. 2, pp. 812–820, 2016.

[68] R. Farnese, W. Melo, and A. Veiga, *JMetrics: Java Metrics Extractor. Oracle Consulting Services*. Brazil, 1999.
[69] 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.
[70] J.-J. M. Tool, "Java measurement tool," Mar 2020. [Online]. Available: www2.informatik.hu-berlin.de/swt/intkoop/jcse/tools/jmt.html
[71] K. T. Website, "Power software," Mar 2020. [Online]. Available: http://www.powersoftware.com/
[72] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," *Journal of Systems and Software*, vol. 51, no. 3, pp. 245–273, 2000.
[73] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial use of metrics for object-oriented software: An exploratory analysis," *IEEE Trans. Softw. Eng.*, vol. 24, no. 8, p. 629639, Aug. 1998. [Online]. Available: https://doi.org/10.1109/32.707698
[74] Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen, "An empirical study on object-oriented metrics," in *Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403)*, Nov 1999, pp. 242–249.
[75] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: implications for software defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, April 2003.
[76] A. Srivastava, J. Thiagarajan, and C. Schertz, "Efficient integration testing using dependency analysis," Technical Report MSR-TR-2005-94, Microsoft Research, Tech. Rep., 2005.
[77] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 531–540.
[78] mccabe software, "mccabe software," Mar 2020. [Online]. Available: http://www.mccabe.com/iq.htm
[79] M. Hericko, I. Rozman, R. V. Horvat, T. Domajnko, and J. Gyorkos, "Oo metrics data gathering environment," in *Proceedings. Technology of Object-Oriented Languages. TOOLS 24 (Cat. No.97TB100240)*, 1997, pp. 80–85.
[80] J. S. Alghamdi, R. A. Rufai, and S. M. Khan, "Oometer: a software quality assurance tool," in *Ninth European Conference on Software Maintenance and Reengineering*, 2005, pp. 190–191.
[81] R. Software, "Rational rose," Mar 2020. [Online]. Available: https://www.ibm.com/products/software
[82] Sonargraph, "hello2morrow," Mar 2020. [Online]. Available: https://www.hello2morrow.com/products/sonargraph/explorer
[83] R. Roveda, "Identifying and evaluating software architecture erosion," 2018.
[84] S. development laboratory., "Software development laboratory," Mar 2020. [Online]. Available: http://www.sdml.info/projects/srcml/
[85] S. Kpodjedo, F. Ricca, G. Antoniol, and P. Galinier, "Evolution and search based metrics to improve defects prediction," in *2009 1st International Symposium on Search Based Software Engineering*, May 2009, pp. 23–32.
[86] STAN, "stan," Mar 2020. [Online]. Available: http://stan4j.com/
[87] G. Bucci, F. Fioravanti, P. Nesi, and S. Perlini, "Metrics and tool for system assessment," in *Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems (Cat. No.98EX193)*, 1998, pp. 36–46.
[88] F. Fioravanti, P. Nesi, and S. Perlini, "A tool for process and product assessment of c++ applications," in *Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering*, 1998, pp. 89–95.
[89] S. P. Borgatti, M. G. Everett, and L. C. Freeman, "Ucinet for windows: Software for social network analysis," 2002.
[90] S. T. Inc., "Scitools maintenance, metrics and documentation tools for ada, c, c++, java and fortran," Mar 2020. [Online]. Available: http://www.scitools.com/
[91] M. English, C. Exton, I. Rigon, and B. Cleary, "Fault detection and prediction in an open-source software project," in *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, ser. PROMISE '09. New York, NY, USA: ACM, 2009, pp. 17:1–17:11.
[92] M. Scotto, A. Sillitti, G. Succi, and T. Vernazza, "A relational approach to software metrics," in *Proceedings of the 2004 ACM Symposium on Applied Computing*, ser. SAC 04. New York, NY, USA: Association for Computing Machinery, 2004, p. 15361540. [Online]. Available: https://doi.org/10.1145/967900.968207
[93] R. Malhotra, A. Kaur, and Y. Singh, "Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines," *International Journal of System Assurance Engineering and Management*, vol. 1, no. 3, pp. 269–281, 2010.
[94] A. Mockus, "Amassing and indexing a large sample of version control systems: Towards the census of public source code history," in *2009 6th IEEE International Working Conference on Mining Software Repositories*, 2009, pp. 11–20.
[95] "The mozilla homepage," http://www.mozilla.org, accessed: 2020-04-05.
[96] 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.
[97] 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
[98] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *2009 IEEE 31st International Conference on Software Engineering*, 2009, pp. 298–308.
[99] A. Zeller, *Can We Trust Software Repositories?* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 209–215. [Online]. Available: https://doi.org/10.1007/978-3-642-37395-4_14
[100] S. Kim, H. Zhang, R. Wu, and L. Gong, "Dealing with noise in defect prediction," in *2011 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 481–490.
[101] T. Illes-Seifert and B. Paech, "Exploring the relationship of history characteristics and defect count: An empirical study," in *Proceedings of the 2008 Workshop on Defects in Large Software Systems*, ser. DEFECTS 08. New York, NY, USA: Association for Computing Machinery, 2008, p. 1115. [Online]. Available: https://doi.org/10.1145/1390817.1390821
[102] bugzilla, "Bugzilla for mozilla," Mar 2020. [Online]. Available: http://bugzilla.mozilla.org
[103] T. Zimmermann and N. Nagappan, "Predicting defects with program dependencies," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, Oct 2009, pp. 435–438.
[104] 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.

[105] N. Seliya and T. M. Khoshgoftaar, "Software quality estimation with limited fault data: a semi-supervised learning perspective," *Software Quality Journal*, vol. 15, no. 3, pp. 327–344, 2007.

[106] 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.

[107] 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.

[108] 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.

[109] 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.

[110] 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.

[111] 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.

[112] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 432–441.

[113] 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.

[114] 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.

[115] 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.

[116] 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.

[117] 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*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 61:1–61:11.

[118] G. Canfora, A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, "Multi-objective cross-project defect prediction," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, March 2013, pp. 252–261.

[119] A. Panichella, R. Oliveto, and A. D. Lucia, "Cross-project defect prediction models: L'union fait la force," in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, Feb 2014, pp. 164–173.

[120] 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.

[121] 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.

[122] 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.