On Software Engineering Repositories and Their Open Problems

Daniel Rodriguez  
University of Alcala  
Alcala de Henares, Spain  
daniel.rodriguez@uah.es

Israel Herraiz  
Technical University of Madrid  
Madrid, Spain  
israel.herraiz@upm.es

Rachel Harrison  
Oxford Brookes University  
Oxford, United Kingdom  
rachel.harrison@brookes.ac.uk

Abstract—In the last decade, a large number of software repositories have been created for different purposes. In this paper we present a survey of the publicly available repositories and classify the most common ones as well as discussing the problems faced by researchers when applying machine learning or statistical techniques to them.

Keywords—quality; software engineering repositories; prepro- 
cessing software engineering data; data quality

I. INTRODUCTION

Many research studies in the field of Empirical Software Engineering are based on a few case studies or small samples. For instance, Mockus et al. [1] found that free / open source software contained fewer defects than proprietary systems, basing their conclusions on two cases of open source projects (Apache and Mozilla). Another example is the distribution of bugs in software, which was recently found to be a Weibull distribution [2] based on some releases of a single case study (the Eclipse IDE). But this result was refuted by a report with contradictory results, stating that the distribution of bugs can also be described using other statistical models [3].

Empirical research that is based on small datasets will have to refute contradictory results because of lack of generalisation. However, gathering a large amount of software and data for empirical studies can be a cumbersome task, prone to the introduction of unintentional errors, and potentially causing more problems than they solve.

The popularization and rise of the free / open source software development phenomenon has made available vast amounts of data which are useful for research purposes. Thus, we can find several opportunities in the research community to obtain data for large samples of software projects, and in an integrated and structured manner, so these repositories can be easily queried to extract information. Even some closed repositories with more specialized information have appeared.

These repositories can be applied in any area of the Empirical Software Engineering field. We highlight the case of Search Based Software Engineering (SBSE), because it is particularly suitable for large amounts of data [4].

SBSE deals with research into search and metaheuris- 
tic techniques in software engineering and has become an important area of research. Many SBSE problems are composed of one or more fitness functions that evaluate a search space, which can be generated while searching for the solution or from repositories forming a combinatorial problem from dataset attributes.

Another source of data is simulation, for example, using System Dynamics. Shepperd and Kaoda [5] use simulated data to compare effort estimation methods and it can share some of the problems highlighted later on. We however do not review the technique and specific issues of this approach, and focus on the case of reusable research datasets.

In this position paper we want to provide a preliminary review and classification/characterization of currently available repositories as well as to highlight the most common problems that users and researchers face when dealing with such repositories.

II. SOFTWARE ENGINEERING REPOSITORIES

Software projects leave a trail in different kinds of repositories, and this trail can be used to reconstruct the history of the project, and to study the software development and maintenance processes. We classify this trail in the following categories:

- Source code
  This is the most obvious product of the a software project. Source code can be studied to measure its properties, such as size or complexity.
- Source Code Management Systems (SCM)
  SCM repositories make it possible to store all the changes that the different source code files undergo during the project. Also, SCM systems allow for work to be done in parallel by different developers over the same source code tree. Every change recorded in the system is accompanied with meta-information (author, date, reason for the change, etc) that can be used for research purposes.
- Issue tracking systems
  Bugs, defects and user requests are managed in issue tracking systems, where users and developers can fill tickets with a description of a defect found, or a desired new functionality. All the changes to the ticket are
recorded in the system, and most of the systems also record the comments and communications among all the users and developers implied in the task.

- Messages between developers and users
  In the case of free / open source software, the projects are open to the world, and the messages are archived in the form of mailing lists, which can also be mined for research purposes. There are also some other open message systems, such as IRC or forums. Other projects which are developed in public can also store messages, but it is unusual to have that information for research purposes.

- Meta-data about the projects
  As well as the low level information of the software processes, we can also find meta-data about the software projects which can be useful for research. This meta-data may include intended-audience, programming language, domain of application, license (in the case of open source), etc.

- Usage data
  In the case of the user side, the trail that projects leave is virtually invisible. There are statistics about software downloads on the Internet, but that is not the only way users get their software. Some of the research datasets we describe in this paper include information about usage data, which is recorded thanks to the collaboration of users.

III. RESEARCH DATASETS

As stated previously there is a large number of repositories that have been created in the last decade that allow researchers to study different aspects within the software engineering field with statistical or data mining techniques. In this paper we analyze the following repositories:

- FLOSSMole [6]
  http://flossmole.org/
- FLOSSMetrics [7]:
  http://flossmetrics.org/
- PROMISE (PRedictOr Models In Software Engineering) [8]:
  http://promisedata.org/
- Qualitas Corpus (QC) [9]:
  http://qualitascorpus.com/
- Sourcerer Project [10]:
  http://sourcerer.ics.uci.edu/
- Ultimate Debian Database (UDD) [11]
  http://udd.debian.org/
- Bug Prediction Dataset (BPD) [12], [13]:
  http://bug.inf.usi.ch/
- The International Software Benchmarking Standards Group (ISBSG) [14]
  http://www.isbsg.org/
- Eclipse Bug Data (EBD) [11], [15]:
  http://www.st.cs.uni-saarland.de/sotiveo/bug-data/eclipse/
- Software-artifact Infrastructure Repository (SIR) [16]
  http://sir.uni1.edu
- ohloh [17]
  http://www.ohloh.net/
- SourceForge Research Data Archive (SRDA) [18]
  http://zerlot.cse.nd.edu/

IV. CLASSIFICATION OF THE DATASETS

We can classify the repositories into several orthogonal dimensions:

- Type of information stored in the repositories:
  - Meta-information about the project itself and the people that participated.
  - Low-level information
    * Mailing Lists (ML)
    * Bugs Tracking Systems (BTS) or Project Tracker System (PTS)
    * Software Configuration Management Systems (SCM)
  - Processed information. For example Project management information about the effort estimation and cost of the project.

- Whether the repository is public or not

- Single project vs. multiprojects. Whether the repository contains information of a single project with multiples versions or multiples projects and/or versions.

- Type of content, open source or industrial projects

- Format in which the information is stored. The repositories analysed here provide the information using different formats or technologies for accessing the information:
  - Text. It can be just plain text, CSV (Comma Separated Values) files, Attribute-Relation File Format (ARFF) [19] or its variants
  - Through databases. Downloading dumps of the database.
  - Remote access such as APIs of Web services or REST.

V. QUALITY ISSUES / OPEN PROBLEMS

We can also classify the problems related to the extraction of information or from the actual data stored in the repositories.

A. Problems generated from extracting the information

Robles et al. [20] describe the processes and tools to extract information needed to analyse software repositories. Although the process is quite similar to the general process of data mining described by Fayyad et al. [21], it has its own characteristics and difficulties. There is large variability in
the formats and tools needed, standards, etc. that make the data gathering process a very labour intensive one. Another example is the mining of textual data to deal with bugs for classification, clustering, etc. This is a difficult task even with human intervention because change requests and incident reports are often mixed together in the BTS or PTS.

### B. Replicability

Replicability is one of the main reasons to adopt open repositories [22]. Kitchenham [23] also discusses the risk of replicating experiments without using the original sources. It is a well known fact that in the data mining process, one of the hardest tasks is to preprocess the data. However, trusting the preprocessed data from others is a poisoned chalice. For example, Shepperd has reported differences between using an original dataset or a preprocessed one downloaded from the PROMISE repository [8]. Among the repositories discussed, EBD not only contains the data but also the necessary scripts to replicate the study.

### C. Data quality problems related to machine learning

From the statistical and data mining point of view, we face many of the generic problems we discuss in this section. Therefore, in addition to specific tool issues (e.g., [24]) we need to be aware of many of the statistical and data mining problems we may face when dealing with software engineering repositories.

- **Outliers.**
  Although this statistical problem is well known in the literature, it is not always properly reported for example in many estimation studies as stated by Turhan [25].

- **Missing values and inconsistencies.**
  Some of the repositories such as the ISBSG, are composed of a large number of attributes, however, many of those attributes are missing values that need to be discarded in order to apply machine learning algorithms. There are also inconsistencies in the way information is stored [26]. In this particular dataset, cleaning inconsistencies (e.g., languages classified as 3GL or 4GL, Cobol 2 or Cobol II) can be risky.

- **Redundant and irrelevant attributes and instances.**
  It is also well known that the existence of irrelevant and redundant features in the datasets has a negative impact in most data mining algorithms, which assume a certain level of balance between the class attributes. Feature Selection has been applied and studied by the software engineering community, not so much instance selection which needs further research (a few exceptions for effort estimation include [27], [28]). It is known, however, that feature selection algorithms do not perform well with imbalanced datasets, resulting in a selection of metrics that are not adequate for the learning algorithms. This problem can happen in most effort estimation or defect prediction datasets. For example, the ISBSG that has over 60 attributes most of them are irrelevant or the 8000 repeated rows in JM1 from NASA’s defect prediction datasets in PROMISE. Also the defect prediction datasets such the EB data are highly unbalanced. Some further research into robust algorithms such as Subgroup Discovery techniques is also needed [29] or weighting of attributes and instances.

- **Overlapping or class separability.**
  When dealing with classification, we may also face the problem of overlapping between classes in which a region of the data space contains samples from different values for the class. We have found that many samples from the NASA dataset contained in the PROMISE repository are contradictory or inconsistent, many in-

---

### Table 1

<table>
<thead>
<tr>
<th>Repository</th>
<th>Meta-info</th>
<th>Low-level Info</th>
<th>Public?</th>
<th>Single vs. Multi</th>
<th>OSS</th>
<th>Format/Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOSSMole</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Multi</td>
<td>Yes</td>
<td>DB dumps, text, DB access</td>
</tr>
<tr>
<td>FLOSSMetrics</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Multi</td>
<td>Yes</td>
<td>DB dumps, web service, web</td>
</tr>
<tr>
<td>PROMISE</td>
<td>Yes</td>
<td>Some datasets</td>
<td>Yes</td>
<td>Multi</td>
<td>Yes</td>
<td>Mostly ARFF</td>
</tr>
<tr>
<td>QC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Multi</td>
<td>Yes</td>
<td>CSV, source code, JAR</td>
</tr>
<tr>
<td>SourceR</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Single (Debian)</td>
<td>Yes</td>
<td>DB dump</td>
</tr>
<tr>
<td>UDD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>CSV</td>
</tr>
<tr>
<td>BPD</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>CSV</td>
</tr>
<tr>
<td>ISBSG</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Multi</td>
<td>No</td>
<td>Spreadsheet</td>
</tr>
<tr>
<td>EBD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Single (Eclipse)</td>
<td>Yes</td>
<td>ARFF, CSV</td>
</tr>
<tr>
<td>SIR</td>
<td>No</td>
<td>Actual code</td>
<td>Needs registration</td>
<td>Multi</td>
<td>Yes</td>
<td>C/Java/C#</td>
</tr>
<tr>
<td>ohloh</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Needs registration, only research</td>
<td>Yes</td>
<td>Web service (limited)</td>
</tr>
<tr>
<td>SRDA</td>
<td>Yes (from SFNet)</td>
<td>Yes</td>
<td>Yes</td>
<td>Multi</td>
<td>Yes</td>
<td>BD dumps</td>
</tr>
</tbody>
</table>
stances have the same values for all attributes with the exception of the class, making the induction of good predictive models difficult.

- Data shifting.
  The data shift problem happens when the test data distribution differs from the training distribution. Turhan [25] discusses the dataset shift problem in software engineering (effort estimation and defect prediction). It is customary in data mining, to perform the evaluation using cross-validation, i.e., divide the dataset into \( k \)-folds for training and testing and report the averages of the \( k \) folds. This problem can easily happen when we are dealing with small datasets [30]. Also when we are dealing with small datasets, it can happen that the number of instances that remain in the training dataset is skewed. Many software effort estimation datasets are very small (around 20 effort estimation datasets contained in PROMISE repository contain just over a dozen samples, e.g., the Kemerer or Telecom datasets).

- Imbalance.
  This happens when samples of some classes vastly outnumber the cases of other classes. Under this situation, when the imbalanced data is not considered, many learning algorithms generate distorted models for which (i) the impact of some factors can be hidden and (ii) the prediction accuracy can be misleading. Although this is a well-known problem in the data mining community, this problem has not been addressed in detail by the software engineering community. This is typically addressed by preprocessing the datasets with sampling techniques or considering cost in the data mining algorithms (making the algorithms more robust). This problem happens in many of the defect prediction datasets (e.g., the PROMISE repository has around 60 defect prediction datasets). The previous problems, redundant and irrelevant attributes, overlapping, data shifting and small datasets are made worse when datasets are imbalanced [31].

- Metrics.
  In relation to the measurements, either from the social network data, mailing lists or code, there can be differences depending on the tool used in those repositories that contain source code such FLOSSMetrics, EBD, or BPD. For example, Lincke et al. [32] report on large differences in metrics collected from the code depending on the tool used.

- Evaluation metrics and the evaluation of models.
  For example, Shepperd and MacDonell [33] report on the the use and abuse of using MMRE (Mean Magnitude of Relative Error) when dealing with effort estimation. Despite the fact that MMRE has been known to be biased and favors underestimation, perhaps because it is easy to apply, it has been used to wrongly validate and compare different estimation methods or models. Furthermore, as such metrics can be used as fitness functions in metaheuristic algorithms [34], the solutions obtained may be suboptimal.

VI. Conclusions

In this position paper we have discussed the current data repositories that are available for Software Engineering research. We classified them and discussed some common problems faced when extracting information from them. We have also discussed data related problems when applying machine learning techniques. Although some of the problems such as outliers or noise have been extensively studied in software engineering, others need further research, in particular, imbalance and data shifting from the machine learning point of view and replicability in general, providing not only the data but also the tools to replicate the empirical work.

References


