Bug Report Severity Level Prediction in Free Libre/Open Source Software: A State of Art Survey

Author1 Affiliation1 Author2 Affiliation2 Author2 Affiliation2

Abstract

(i)Establishing a territory. (ii)Indicating a gap. (iii)Announcing present research. (iv)Announce principal findings.

Index Terms

software maintenance; bug tracking systems; bug reports; severity level prediction; software repositories.

I. Introduction

1. show the general area is important, problematic, or relevant in some way (motivations)
2. introduce and review items of previous research in the area
3. Indicate a gap in the previous research, or extend previous knowledge in some way
4. Outline purposes or state the nature of present research
5. Announce principal findings (contributions) and its importance
6. indicate the structure of the research paper

II. BACKGROUND

This section briefly comments of basic concepts necessary to understand this research area

A. Related Work

Introduce and review related works in the area:

- Cavalcanti et al. [1]

III. Research Method

This section describes the research method used in this mapping study for identifying and analyzing relevant papers. It was based on the software engineering systematic literature review guidelines and recommendations proposed by Kitchenham et al. [2], Brhel et al. [3], Souza et al. [4], and Petersen et al. [5]. They recommend a process with three main steps regarding with review: planning, conducting, and reporting. In the first step, it should be specified a review protocol, which that contains a set of research questions, inclusion and exclusion criteria, sources of papers, search string, and mapping procedures. In the second step, it was selected and retrieved papers to use as basis for data extraction. . Finally, in the last step, the results used to address the systematic mapping research questions defined previously was written up.

A. Research questions

The main mapping study's goal is to provide a current status view of the research on bug report severity prediction in FLOSS. To ensure an unbiased selection process, we defined previously the following research questions to be addressed, as well as the rationale for considering them in this mapping study.

RQ$_1$. When and where have the studies been published? This research question gives to researcher a perception whether the topic of this mapping study seems to be broad and new, provinding an overview about where and when papers were published.

RQ$_2$. What FLOSS are the most used as experiment target for bug report severity prediction? This research question points out the FLOSS used in bug report severity. This overview can be useful for researchers that intend to accomplish new initiatives in this area. It can be also useful to motivate adoption of new FLOSS in future researches to bridge existing gaps.

RQ$_3$. Has bug report severity prediction been more commonly treated as a two-label or multi-label classification problem? This research question investigates whether bug report severity prediction has been commonly treated as a fine-grained label (i.e., multi-labels) or as a coarse-grained label (i.e., binary labels) classification problem. This is essential to comprehend clearly each solution provided for each classification problem type.

RQ$_4$. What are the most common features used bug report severity prediction? This question identifies features used for bug report severity prediction in FLOSS. It can help to map which features considered more effective in this more effective in this prediction.

RQ$_5$. What are the most common features selection methods used for bug report severity prediction? This question complements the previous one, looking for feature selection methods employed in the studies for bug report severity prediction in FLOSS. This is important to point out what features selection techniques have been considered more suitable for this classification type.

RQ$_6$. What are the most used text mining or information retrieval methods for bug report severity prediction? This research question indicates the main text mining approaches used to extract features from textual fields of bug reports. It can support researchers to select text mining methods more suitable to be applied in bug report severity prediction or similar problems.

RQ$_7$. What are the most used machine learning algorithms for bug report severity prediction? This research question aims to identify the main ML algorithms adopted bug report severity prediction in FLOSS. What can be useful for any researcher that aim to accomplish further initiatives in this area, as well as to guide him investigating others algorithms to improve current results.

RQ$_8$. What are the measures typically used to evaluate ML algorithms performance for bug report severity prediction? This research question aims to find out which measures are used to evaluate ML algorithms performance in selected papers. It can be helpful to identify the more effective measures to evaluate ML algorithms performance for for bug report severity prediction.

RQ$_9$. Which sampling techniques were employed to improve ML algorithms performance for bug report severity prediction? This question complements the previous one, investigating the sampling techniques, which were used in selected papers to generate more reliable and accurate ML models. It can be useful to analyze sampling techniques suitability to improve the machine learning algorithms performance for bug report severity prediction.

RQ$_{10}$. What statistical tests was used to compare the performance between two or more ML algorithms for bug report severity prediction? The answer to this research question can be useful to identify what most statistical tests were used in bug report severity prediction context. In addition, to grasp how the comparison between two or more ML algorithms performance using statistical significance was made.

RQ$_{11}$. What software tools were used to run bug report severity prediction experiments? This research question highlights the software tools most used to run experiments for bug reports severity prediction in FLOSS projects. This is useful to provide information for researchers and practitioners about technologies that could be used in their own experiments.

RQ$_{12}$. What solution types were proposed for bug report severity prediction problem? This research question examines whether the most common solutions proposed in the selected studies are online or off-line. This important to separate clearly the solutions that can be applied only in an experimental environment (off-line) from those also can be deployed in a real scenario (on-line).

B. Study selection

This section describes the selection process carried out through the mapping study. It comprises four steps: (i) terms and search string; (ii) sources for searching; (iii) inclusion and exclusion criteria; and (iv) how to store data.

1) Terms and search string: The base string was constructed from three search main terms: open source project, bug report and severity predict. To build the search string, terms were combined with "AND" connectors. The search string syntax was adapted according particularities of each source (e.g. wildcars, connectors, apostrophes, quotation marks, and etc) before it has been applied on three meta-data papers: title, abstract, and keywords. Table I exhibits terms and closing search string used in this mapping study.

TABLE I: Mapping study search string.

Areas	Search terms
Bug report	"bug report"
Open source	"open source software"
Severity prediction	"severity predict"
Search string:	"open source software" AND "severity predict" AND "bug report"

2) Sources: We selected four electronic databases and one popular search engines recommended by Kitchenham et al. [2]. Table II shows each selected sources, as well as the search date, and the period covered by the search.

3) Inclusion and exclusion criteria: The inclusion criteria below let look for papers on the existing literature.

IC$_1$. The study discusses bug report severity prediction in FLOSS projects.

On the contrary, the following exclusion criteria let us exclude all papers that met any of them.

TABLE II: Search sources.

Source	Electronic address	Type	Search Date	Years covered
ACM	http://dl.acm.og	Digital library	Dec,18	2010 - 2017
IEEE	http://ieeexplore.ieee.org	Digital library	Dec,18	2010 - 2017
Google Scholar	http://www.scholar.google.com	Search engine	Dec,18	2010 - 2017
Science Direct	http://www.sciencedirect.com	Digital library	Dec,18	2010 - 2017
SpringLink	http://www.springerlink.com	Digital library	Dec,18	2010 - 2017

EC$_1$. The study does not have an abstract;
EC$_2$. The study is just published as an abstract;
EC$_3$. The study is written in a language other than English;
EC$_4$. The study is not a primary study (e.g., keynotes);
EC$_5$. The study is not accessible on the Web;

4) Data storage: The data extracted in the searching phase was stored into a spreadsheet that recorded all relevant details from selected papers . This spreadsheet supported classification and analysis procedures throughout this mapping study.

5) Assessment: According to Kitchenham et al. [2], the consistency of the protocol utilized in a mapping study should be reviewed to confirm that: "the search strings, constructed interactively derived from the research questions"; "the data to be extracted will properly address the research question(s)"; "the data analysis procedure is appropriate to answer the research questions". In this research, first author is a PhD candidate running the experiments. The review process was conducted by the second and third authors.

C. Data extraction and synthesis

Figure 1 depicts four stages of selection process carried out in current mapping study. In each stage, the sample size was reduced based on the inclusion and exclusion criteria applicable in this stage. In the first stage, relevant papers were retrieved by querying the databases with the search string in Table I. All database queries were issued in December 2017. This yielded a total of 54 initial papers: 13 from **IEEE Xplore**, 5 from **Science Direct**, 17 from **ACM Digital Library**, and 19 from **SpringerLink**. After removing four duplicates, we reduced (around 7%) the initial set to 50 publications. In the second stage,inclusion and exclusion criteria were applied over title, abstract, and keywords, reaching a set of 18 papers (reduction around 64%): 32 papers were rejected for not satisfying IC1 (The study discusses severity prediction of bug reports in FLOSS projects). In the third stage, exclusion criteria were applied considering the full text.However, no study was rejected by taking into account these criteria.

In the fourth stage, the Snowballing [2] activity was accomplished, which resulted in more 12 papers. After applying selection criteria over title, abstract, and keywords, 11 papers remained (reduction of 8.3% over the papers selected by snowballing). For these papers, selection criteria were applied considering the full text, and nine papers remained (reduction of approximately 18.2% over the 11 previously selected papers); EC3 criterion eliminated one paper (the study is written in a language other than English); EC6 eliminated one more paper (the study is inaccessible on the Web), and another one was rejected for not satisfying IC1.

The selection phase ended up with 27 papers to be analyzed (18 from the sources plus 9 from snowballing). Table III shows the bibliographic reference of selected papers and an identifier (#id) for each paper. Throughout the remainder of this text, these identifiers will be used to refer to the corresponding paper.

D. Classification scheme

Petersen et al. [5] suggest the definition of a classification scheme for conducting a systematic mapping analysis. The categories defined for this mapping study was based in two approaches: (i) on categories available in the literature and (ii) taking into account the selected papers. The next sections outline the categories that will be used in this mapping study.

1) FLOSS software type (RQ_2): This classification organizes the FLOSS used as targets in selected papers experiments by software type. Based on taxonomy suggested by Pressman [?], it was considered three categories:

a. **Application software**: a computer program or group of computer programs designed to solve a specific problem or business need for end users.
b. **System software**: a collection of computer programs (operating systems and utilities) required to run and maintain a computer system.
c. **Programming tools**: computer programs that aid software developers to create, debug, maintain or perform any development-specific task.

2) ML classification problem (RQ_3):

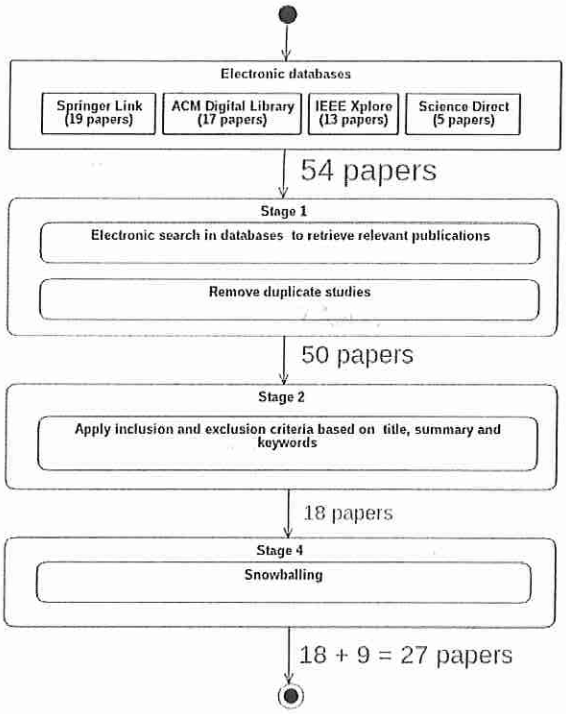


Fig. 1: Study selection phases diagram.

3) Feature data types (RQ_4): This classification groups the features used for bug reports severity prediction into well-known three categories:

a. **Qualitative**: data that can be arranged into categories based on specific characteristics. It are cannot be expressed numerically.
b. **Quantitative discrete**: data that contains a finite or a infinite number of values that can be counted.
c. **Quantitative continuous**: data that contains a infinite number of values that can be measured.

4) Feature selection methods (RQ_5): This classification organizes the feature selection methods used for bug reports severity prediction into three categories, according to taxonomy described in Guyon and Elisseeff [6]:

a. **Filter**: methods that apply a statistical measure to assign a scoring to each feature. Features are ranked by this score which is used to keep or to remove one from a dataset.
b. **Wrapper**: methods that consider the features selection as a search problem, where different combinations are prepared, evaluated and compared to other combinations.
c. **Embedded**: methods that learn which features best contribute to the accuracy of the model while the model is being created.

5) Text mining feature representations (RQ_6): This classification organizes the text mining features representations techniques used for bug report severity prediction in four categories. Following, such categories employed in this mapping study according to [7]:

a. Character: the individual component-level letters, numerals, special characters and spaces are the building blocks of higher-level semantic features such as words, terms and concepts.
b. Word: specific words selected directly from a "native" document are at what might be described as the basic level of semantic richness.
c. Term: terms are single words an multiword phrases selected directly from corpus of a native document by means of term-extraction methodologies.
d. Concept: Concepts are features generated for a document by means of manual, statistical, rule-based or hybrid categorization methodologies.

6) ML algorithms types (RQ_7): This classification groups ML algorithms used in bug report severity prediction in five categories. The categories were based on taxonomy proposed by Facelli et al. [8]:

a. **Distance-based**: algorithms that use the proximity between data to generate their predictions.
b. **Probabilistic-based**: algorithms that make their predictions based on the Bayes's theorem.

TABLE III: Selected studies

#ID,	Bibliographic reference
#1	A. Lamkanfi, S. Demeyer, E. Giger and B. Goethals, "Predicting the severity of a reported bug," 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), Cape Town, 2010, pp. 1-10.
#2	A. Lamkanfi, S. Demeyer, Q. D. Soetens and T. Verdonck, "Comparing Mining Algorithms for Predicting the Severity of a Reported Bug", 2011 15th European Conference on Software Maintenance and Reengineering, Oldenburg, 2011, pp. 249-258.
#3	Y. Tian, D. Lo and C. Sun, "Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction," 2012 19th Working Conference on Reverse Engineering, Kingston, ON, 2012, pp. 215-224.
#4	C. Z. Yang, C. C. Hou, W. C. Kao and I. X. Chen, "An Empirical Study on Improving Severity Prediction of Defect Reports Using Feature Selection," 2012 19th Asia-Pacific Software Engineering Conference, Hong Kong, 2012, pp. 240-249.
#5	Chaturvedi, K. K. and V.B. Singh. "An Empirical Comparison of Machine Learning Techniques in Predicting the Bug Severity of Open and Closed Source Projects." IJOSSP 4.2 (2012): 32-59.
#6	G. Yang, T. Zhang and B. Lee, "Towards Semi-automatic Bug Triage and Severity Prediction Based on Topic Model and Multi-feature of Bug Reports," 2014 IEEE 38th Annual Computer Software and Applications Conference, Vasteras, 2014, pp. 97-106.
#7	C. Z. Yang, K. Y. Chen, W. C. Kao and C. C. Yang, "Improving severity prediction on software bug reports using quality indicators," 2014 IEEE 5th International Conference on Software Engineering and Service Science, Beijing, 2014, pp. 216-219.
#8	Harold Valdivia Garcia and Emad Shihab. 2014. "Characterizing and predicting blocking bugs in open source projects". In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014). ACM, New York, NY, USA, 72-81.
#9	Sharma M., Kumari M., Singh R.K., Singh V.B. (2014) "Multiattribute Based Machine Learning Models for Severity Prediction in Cross Project Context". In: Murgante B. et al. (eds) Computational Science and Its Applications – ICCSA 2014. ICCSA 2014. Lecture Notes in Computer Science, vol 8583. Springer, Cham.
#10	N. K. S. Roy and B. Rossi, "Towards an Improvement of Bug Severity Classification," 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, Verona, 2014, pp. 269-276.
#11	Ripon K. Saha, Julia Lawall, Sarfraz Khurshid, and Dewayne E. Perry. 2015. "Are these bugs really "normal"?". In Proceedings of the 12th Working Conference on Mining Software Repositories (MSR '15). IEEE Press, Piscataway, NJ, USA, 258-268.
#12	Tao Zhang, Geunseok Yang, Byungjeong Lee, and Alvin T. S. Chan. 2015. "Predicting severity of bug report by mining bug repository with concept profile". In Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15). ACM, New York, NY, USA, 1553-1558.
#13	G. Sharma, S. Sharma, S. Gujral, "A Novel Way of Assessing Software Bug Severity Using Dictionary of Critical Terms", Procedia Computer Science, pp. 632-639, 2015.
#14	X. Xia, D. Lo, E. Shihab, X. Wang, X. Yang, "Elblocker: Predicting blocking bugs with ensemble imbalance learning", Information and Software Technology, 2015.
#15	S. Gujral, G. Sharma, S. Sharma and Diksha, "Classifying bug severity using dictionary based approach," 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), Noida, 2015, pp. 599-602.
#16	M. N. Pushpalatha and M. Mrunalini, "Predicting the severity of bug reports using classification algorithms," 2016 International Conference on Circuits, Controls, Communications and Computing (I4C), Bangalore, 2016, pp. 1-4.
#17	A. F. Otoom, D. Al-Shdaifat, M. Hammad and E. E. Abdallah, "Severity prediction of software bugs," 2016 7th International Conference on Information and Communication Systems (ICICS), Irbid, 2016, pp. 92-95.
#18	Korosh Koochekian Sabor, Mohammad Hamdaqa, and Abdelwahab Hamou-Lhadj. 2016. "Automatic prediction of the severity of bugs using stack traces". In Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering (CASCON '16), Blake Jones (Ed.). IBM Corp., Riverton, NJ, USA, 96-105.
#19	Tian, Y., Ali, N., Lo, D. et al. Empir Software Eng (2016) 21: 2298. https://doi.org/10.1007/s10664-015-9409-1
#20	Tao Zhang, Jiachi Chen, Geunseok Yang, Byungjeong Lee, and Xiapu Luo. 2016. "Towards more accurate severity prediction and fixer recommendation of software bugs". J. Syst. Softw. 117, C (July 2016), 166-184.
#21	Kwanghue, J., Amarmend, D., Geunseok, Y., Jung-Won, L., Byungjeong, L.: "Bug severity prediction by classifying normal bugs with text and meta-field information". Adv. Sci. Technol. Lett. 129 (2016). Mechanical Engineering.
#22	T. Choeikiwong, P. Vateekul, "Improve accuracy of defect severity categorization using semi-supervised approach on imbalanced data sets", Proceedings of the International MultiConference of Engineers and Computer Scientists, vol. 1, 2016.
#23	Jin, Kwanghue, Dashbalbar, Amarmend Yang, Geunseok, Lee, Byungjeong, Lee, Jung-Won. (2016). "Improving predictions about bug severity by utilizing bugs classified as normal". Contemporary Engineering Sciences. 9. 933-942. 10.12988/ces.2016.6695.
#24	Jin, K, Lee, E.C., Dashbalbar, A, Lee, J, Lee, B. (2016). "Utilizing feature based classification and textual information of bug reports for severity prediction". 19. 651-659.
#25	Geunseok Yang, Seungsuk Baek, Jung-Won Lee, and Byungjeong Lee. 2017. "Analyzing emotion words to predict severity of software bugs: a case study of open source projects". In Proceedings of the Symposium on Applied Computing (SAC '17). ACM, New York, NY, USA, 1280-1287.
#26	V. B. Singh, Sanjay Misra, and Meera Sharma, J. Info. Know. Mgmt. 16, 1750005 (2017)
#27	N. K. S. Roy and B. Rossi, "Cost-Sensitive Strategies for Data Imbalance in Bug Severity Classification: Experimental Results," 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Vienna, 2017, pp. 426-429.

c. **Searching-based**: algorithms that rely on a searching of a solution in a space to generate their predictions.

d. **Optimization-based**: algorithms which their predictions are based on an optimization function.

e. **Ensemble method**: a set of algorithms which individuals decisions are combined or aggregated to make the prediction.

7) Evaluation measures categories (RQ_8): This classification organizes evaluation measures indicated in selected papers in six categories. It was considered the categories suggested by Japkowicz and Shah [9].

a. **Single class focus**: based on confusion matrix information, this measure focuses on a single class of interest and it is

indicated for deterministic classifiers.

b. **Multi-class focus**: based on confusion matrix information, this measure focuses on all the classes in the domain and it is indicated for deterministic classifiers.

c. **Graphical measure**: based on confusion matrix together information, this measure enable visualization of the classifier performance under different skew ratios and class distribution priors. It is indicated for scoring classifiers.

d. **Summary statistics**: based on confusion matrix together information, this measure enable quantify the comparative analysis between classifiers. It is indicated for scoring classifiers.

e. **Distance/error-measure**: based on confusion matrix together extra information, this measure the distance of an instance's predicted class label to its actual label. It is indicated for continuous and probabilistic classifiers.

f. **Information theoretic measures**: based on confusion matrix together extra information, this measure reward a classifier upon correct classification relative to the (typically empirical) prior on the data. It is indicated for continuous and probabilistic classifiers.

8) Sampling techniques (RQ_9): This classification organizes the sampling techniques used for bug reports severity prediction in three categories. Following, such categories defined in this mapping study according to Japkowicz and Shah [9].

a. **No re-sampling**: techniques that consists of testing the algorithm on a large set of unseen data.

b. **Simple re-sampling**: techniques that tend to use each data point for testing only once.

c. **Multiple re-sampling**: techniques that tend to use each data point for testing more than once.

9) Statistical Tests(RQ_{10}): This classification groups the statistical tests types used for bug reports severity prediction in three categories. Following, such categories employed in this mapping study according to Japkowicz and Shah [9].

a. **Parametric**: make strong assumptions about the distribution of the population

b. **Non-parametric**: do not make strong assumptions about the distribution of the population

c. **Parametric and non-parametric**: both parametric and non-parametric

10) Experiment software tools (RQ_{11}): This classification groups the software tools used in experiments for bug reports severity prediction in two well-know and popular categories:

a. **Free/Libre Open Source Software (FLOSS)**: software that can be freely used, modified, and redistributed.

b. **Closed Source Software (CSS)**: software that is owned by an individual or a company whose source code is not shared with the public for anyone to look at or change.

E. Limitations of this mapping

We need to consider the following common limitations of all mapping studies to interpret adequately the implications of our results:

- **Mapping review completeness can never be guaranteed [3]**: Although in our mapping review, we have observed a strict research protocol to ensure the relatively complete population of the relevant literature, some important papers might be missed.
- **Terminological problems in search string may lead to miss some primary studies [4]**: The terminology we have applied in the database query is normally accepted and used within the scientific community. Nevertheless, different terms may have been used to describe the same relevant information.
- **Subjective evaluation of inclusion and exclusion criteria may cause misinterpretation [3]**. Explicit criteria have been defined for assessing the relevance of selected papers III-B. However, the evaluation was based on the perception and experience of the authors. Different people might have other views regarding the relevance of these papers.

IV. Results

This section summarizes the results of the mapping study. The answers for each research question (RQ1 to RQ12) was presented in tables where each paper is referenced by its id. In addition, extracted data were classified according to criteria defined in Section III.

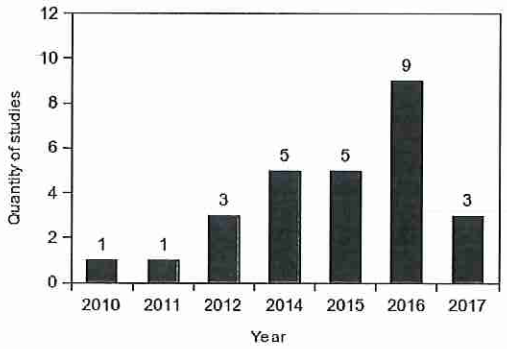
A. When and where have the studies been published? (RQ1)

Figure 2a shows the temporal distribution of papers. Just over 75% (22/27) of primary selected studies were published after 2012. This indicates that research focusing on bug report severity prediction is still recent and relatively stable, with the vast majority of the studies developed in the last four years.

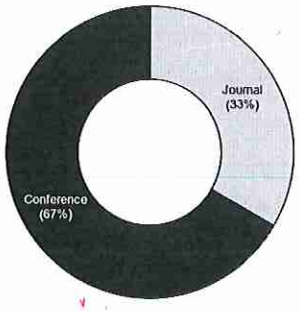
Table IV presents sources of the selected papers, their types, and their identifiers. Although four studies were published in the MSR and two in the SEAA conferences. It is worth observing that 21 out 27 selected papers were published in 21 different venues. This can suggest that currently, there is no particular place for this type of research or there is no well-established forum for debating this topic. Studies in this research seem to be well accepted in venues related to Information Technology, Software Engineering, and Mining Repositories. Figure 2b shows that conferences were preferred rooms to publish 67% of the selected papers, following by journals that were chosen by 33% of studies.

TABLE IV: Papers sources.

Paper source	Category	#ID
ACM Symposium on Applied Computing	Conference	#21
Advanced Science and Technology Letters & Journal	Journal	#10
Asia-Pacific Software Engineering Conference	Conference	#4
Computational Science and Its Applications (ICCSA)	Conference	#9
Computer Software and Applications Conference	Conference	#6
Contemporary Engineering Sciences	Journal	#23
Empirical Software Engineering	Journal	#19
Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)	Conference	#15
Information and Software Technology	Journal	#14
International Conference on Circuits, Controls, Communications and Computing (I4C)	Conference	#16
International Conference on Computer Science and Software Engineering	Conference	#18
International Conference on Information and Communication Systems (ICICS)	Conference	#17
International Conference on Software Engineering and Service Science	Conference	#7
International Information Institute	Journal	#24
International Journal of Open Source Software and Process(IJOSSP)	Journal	#5
International MultiConference of Engineers and Computer Scientists	Conference	#22
Journal of Information & Knowledge Management	Journal	#26
Journal of Systems and Software	Journal	#20
Procedia Computer Science	Journal	#13
Software Engineering and Advanced Applications (SEAA)	Conference	#10, #27
Symposium on Applied Computing (SAC)	Conference	#25
Working Conference on Mining Software Repositories (MSR)	Conference	#1, #2, #8, #11
Working Conference on Reverse Engineering	Conference	#3



(a)



(b)

Fig. 2: (a) Number of papers. (b) Sources of papers.

B. What FLOSS are the most used as experiment target for bug report severity prediction (RQ2)?

Table V details the FLOSS found out after analysis of data gathered throughout this mapping study. The details for each FLOSS include a textual description; a classification, according to Section III-D; a BTS name, which hosted it; and an electronic

address, from where it can be accessed. Table VI shows FLOSS distribution by selected papers. It can be observed that the top five projects were used for at least 11% of papers. Nearly 92% of papers employed Eclipse (25 out of 27), 70% Mozilla (19 out of 27), 18% Openoffice (5 out of 27), 14% Netbeans(4 out of 27) and 11% Gnome(3 out of 27).

TABLE V: FLOSS projects used in studies.

Project	Description	Classification	BTS	Electronic address
Android	Mobile operating system	System software	Google	https://issuetracker.google.com
Chrominum	Web browser	Application software	Google	https://www.chromium.org/issue-tracking
Eclipse	Integrated Development Environment(IDE)	Programming tool	Bugzilla	https://bugs.eclipse.org/bugs/
FreeDesktop	Base platform for desktop for desktop software on Linux and UNIX	Programing Tool	Bugzilla	https://bugs.freedesktop.org/
GCC	C compiler	Programming tool	Bugzilla	https://gcc.gnu.org/bugzilla/
Gnome	Desktop environment based on X Windows System	Application Software	Bugzilla	https://gcc.gnu.org/bugzilla/
Hibernate	Object Relation Mapper(ORM) framework	Programming tool	Jira	https://hibernate.atlassian.net
Jboss	Application server	System software	Jira	https://issues.jboss.org/s
Mongo-db	No-sql database	System software	Jira	https://jira.mongodb.org/
Mozilla	Internet tools	Application software	Bugzilla	https://bugzilla.mozilla.org/
Netbeans	Integrated Development Environment(IDE)	Programming tool	Bugzilla	https://netbeans.org/bugzilla/
OpenOffice	Office suite	Application software	Bugzilla	https://bz.apache.org/ooo/
Spring	JEE Framework	Programming tool	Jira	https://jira.spring.io
WineHQ	Compatibility layer	System software	Bugzilla	https://bugs.winehq.org/

TABLE VI: Projects versus studies.

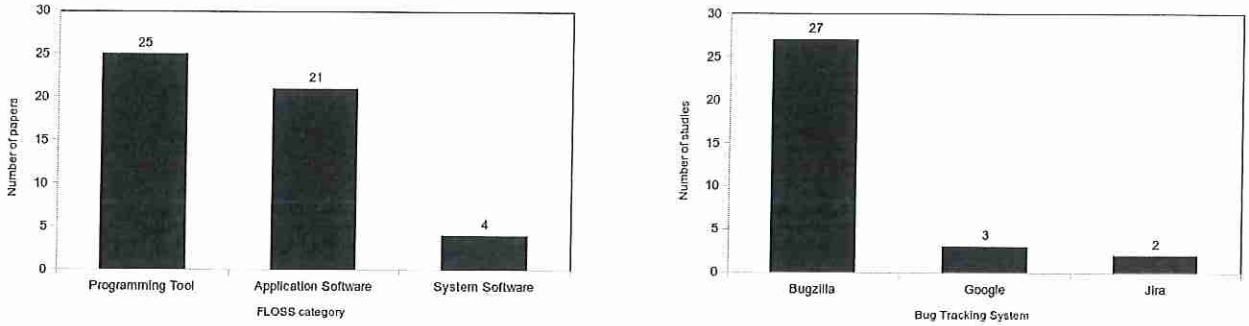
Project	2010	2011	2012	2014	2015	2016	2017	Total (%)
Eclipse	#1	#2	#3, #4, #5	#6, #8, #10	#11, #12, #13, #14, #15	#17, #18, #19, #20, #21, #22, #23, #24	#25, #26, #27	92.6%
Mozilla	#1		#3, #4, #5	#6, #8, #9, #10	#12, #14	#16, #17, #19, #20, #21, #23, #24	#26, #27	70.4%
OpenOffice			#3	#8	#14	#19, #20		18.5%
Netbeans				#6, #8	#14	#20		14.8%
Gnome	#1	#2	#5					11.1%
Chrominum				#8	#14			7.4%
FreeDesktop				#8	#14			7.4%
Android							#25	3.7%
GCC						#20		3.7%
Hibernate							#27	3.7%
JBoss							#25	3.7%
Mongo-db							#27	3.7%
Spring							#27	3.7%
WineHQ						#16		3.7%

The chart in Figure 3a quantifies the number of papers by FLOSS category. It points out that the most FLOSS category investigated in experiments for bug report severity prediction is Programming Tool (25 out of 27), closely followed by Application Software (21 out of 27). FLOSS classified as System Software are used in only 2 out of 27 papers. The chart in Figure 3b quantifies the number of papers by BTS site. It shows up that all (27 out of 27) experiments published in the selected papers has extracted data from at least one FLOSS hosted on a Bugzilla site. In the same chart, It can be observed which only a few of them used FLOSS hosted on Google (3 out of 27) and Jira (2 out of 27).

C. Has the prediction of severity of bug reporting been more commonly treated as a two-label or multi-label classification problem (RQ3)?

D. What are the most common features used bug report severity prediction (RQ4)?

The Table VIII details the features investigated in select papers. The details for each feature include a textual description; a classification, according to Section III-D; whether feature is calculated from others or not; and whether feature is provided



(a) (b)

Fig. 3: (a) Number of papers by FLOSS category. (b) Number of papers by BTS host.

TABLE VII: Classes by papers.

Classes	Use Default Class?	2010	2011	2012	2014	2015	2016	2017	Total(%)
Severe and non-severe	No	#1	#2	#4	#7, #10	#13, #15	#17, #21, #23		37.04%
Multi-labels	No			#3, #5			#16, #18, #19, #20, #22	#26	29.63%
Multi-labels	Yes				#6, #9	#12		#25, #27	18.52%
Blocking and non-blocking	No				#8	#14			7.41%
Severe, non-severe and default label	Yes					#11	#24		7.41%

by Bugzilla, Jira and/or Google. The Table IX presents the feature distribution by papers. It can be observed that summary (19 out of 27 - 81.5%) and description (17 out of 27 - 63.0%) were the most common features used for bug report severity prediction.

The chart in Figure 4 groups features by data type in three categories: qualitative, quantitative discrete and quantitative continuous. It can be observed that most feature data type used for bug report severity prediction was qualitative.

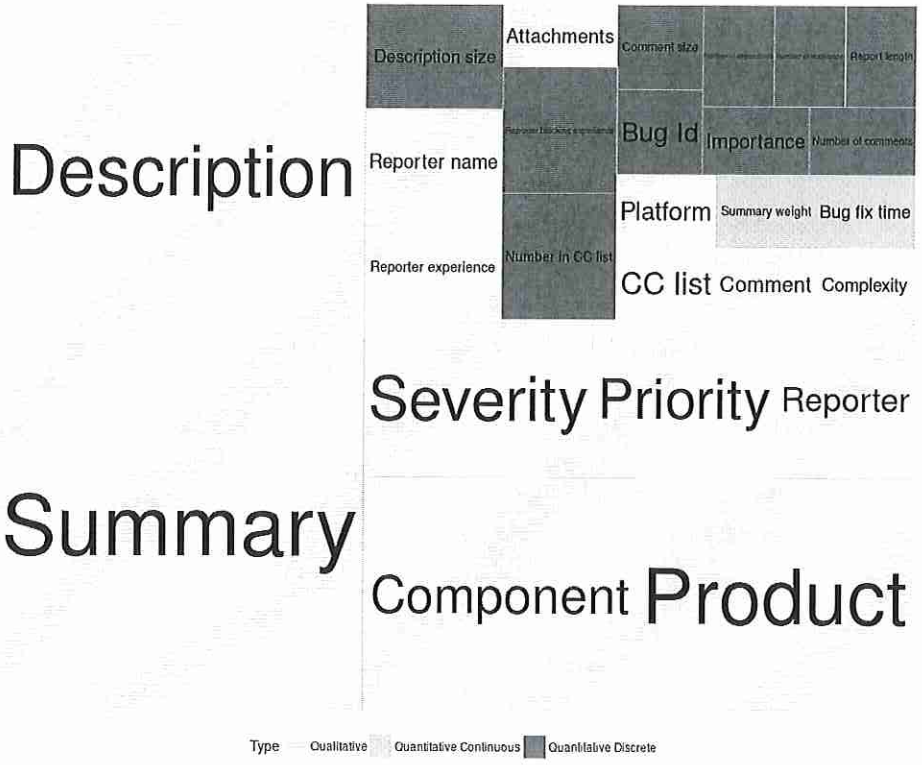


Fig. 4: Papers by feature data type.

TABLE VIII: Features descriptions.

Feature	Description	Classification	Is Calculated?	Bugzilla[1]	Jira[2]	Google[3]
Attachments	Files (e.g. testcases or patches) attached to bugs.	Qualitative	No	Yes	No	Yes
Bug fix time	Time to fix a bug (Last Resolved Time - Opened Time).	Quantitative Continuous	Yes	Yes	Yes	Yes
Bug id	Bug report identifier.	Quantitative Discrete	No	Yes	Yes	Yes
CC list	A list of people who get mail when the bugs changes.	Qualitative	No	Yes	No	Yes
Comment	Textual content appearing in the comments of bug report.	Qualitative	Yes	Yes	Yes	Yes
Comment size	The number of word of all comments of a bug.	Quantitative Discrete	Yes	Yes	Yes	Yes
Complexity	Bug level of complexity based on bug fix time.	Qualitative	Yes	Yes	Yes	Yes
Component	Each product is divided into different components (e.g. Core, Editor, UI, etc).	Qualitative	No	Yes	Yes	Yes
Description	Textual content appearing in the description field of the bug report.	Qualitative	No	Yes	Yes	Yes
Description size	The number of words in the description.	Quantitative Discrete	Yes	Yes	Yes	Yes
Importance	The importance of a bug is a combination of its priority and severity.	Qualitative	No	Yes	No	No
Number in CC list	The number of developers in the CC list of the bug.	Quantitative Discrete	Yes	Yes	No	Yes
Number of comments	Number of comments added to a bug by users.	Quantitative Discrete	Yes	Yes	Yes	Yes
Number of dependents	Number of report bugs dependents	Quantitative Discrete	Yes	Yes	Yes	Yes
Number of duplicates	Number of duplicates of bug report	Quantitative Discrete	Yes	No	No	Yes
Platform	These indicate the computing environment where the bug was found (e.g. Windows, GNU/Linux, Android, etc).	Qualitative	No	Yes	Yes	Yes
Priority	Priority should normally be set by the managers, maintainers or developers who plan to work, not by the one filling the bug or by outside observers.	Qualitative	No	Yes	Yes	Yes[4]
Product	What general "area" the bug belongs to (e.g. Firefox, Thunderbird, Mailer, etc).	Qualitative	No	Yes	Yes	No
Report length	The content length of long description providing debugging information.	Quantitative Discrete	Yes	Yes	Yes	Yes
Reporter	The account of the user who created the bug report.	Qualitative	No	Yes	Yes	Yes
Reporter blocking experience	Counts the number of blocking bugs filed by reporter previous to this bug.	Quantitative Discrete	Yes	Yes	Yes	No
Reporter experience	Counts the number of previous bug reports filed by the reporter.	Quantitative Discrete	Yes	Yes	Yes	No
Reporter name	Name of the developer or user that files the bug	Qualitative	No	Yes	Yes	No
Severity	This indicates how severe the problem is – from blocker ("application unusable") to trivial ("minor cosmetic issue").	Qualitative	No	Yes	Yes	No
Summary	A one-sentence summary of the problem.	Qualitative	No	Yes	Yes	Yes
Summary weight	Calculated using information gain criteria	Quantitative Continuous	Yes	Yes	Yes	Yes

E. What are the most common features selection methods employed for bug report severity prediction (RQ5)?

The Table X shows the methods used in selected papers for bug report severity prediction. It can be observed that: (i) few papers utilized a feature selection method and; (ii) only filter method type was used by them.

F. What are the most used text mining methods for bug report severity prediction (RQ6)?

The Table XI shows text mining methods used in selected papers for bug report severity prediction. It can be observed that the most used text mining method was TF-IDF (9 out of 27 - 33%).

The chart in Figure 5 shows papers by text mining feature model type distribution. It points out that most text mining method used for bug report severity prediction was term model type (12 out of 27 - around 44%).

G. What are the most used machine learning algorithms for bug report severity prediction (RQ7)?

The table XII shows ML algorithms used in selected papers for bug report severity prediction. It can be observed that two most ML algorithms used for bug report severity were KNN and NB (12 out of 27 - around 44%).

The chart in Figure 6 shows ML algorithm types by papers distribution. It points out that most ML algorithm type used for bug report severity prediction was probabilistic based (21 out of 27 - around 77%).

TABLE IX: Features used in selected papers.

Feature	2010	2011	2012	2014	2015	2016	2017	Total (%)
Summary	#1	#2	#3, #4, #5	#6, #7, #9	#11, #13	#16, #17, #20, #21, #23, #24	#25, #26, #27	81.5%
Description	#1		#3	#6, #7, #8, #10	#14, #15	#16, #17, #18, #20, #21, #23, #24	#25, #27	63.0%
Component			#3	#6, #8	#14	#16, #20, #21, #23, #24		33.3%
Product			#3	#6, #8	#14	#16, #20, #21, #23, #24		33.3%
Severity				#8	#14	#21, #23, #24		18.5%
Priority				#6, #8, #9	#14			14.8 %
Reporter						#21, #23, #24		11.1%
Description size				#8	#14			7.4%
Number in CC list				#8	#14			7.4%
Reporter blocking experience				#8	#14			7.4%
Reporter experience				#8	#14			7.4%
Reporter name				#8	#14			7.4%
Attachments				#7				3.7%
Bug fix time				#9				3.7%
Bug Id						#22		3.7%
CC list				#9				3.7%
Comment				#8				3.7%
Comment size					#14			3.7%
Complexity				#9				3.7%
Importance						#16		3.7%
Number of comments				#9				3.7%
Number of dependents				#9				3.7%
Number of duplicates				#9				3.7%
Platform					#14			3.7%
Report length				#7				3.7%
Summary weight				#9				3.7%

TABLE X: Features selection methods used in selected papers.

Method	Classification	2010	2011	2012	2014	2015	2016	2017	Total(%)
Information Gain	Filter			#4, #5		#13	#19	#27	18.52%
Chi-Square	Filter			#4	#10	#13			11.11%
Correlation Coefficient	Filter			#4					3.70%

TABLE XI: Text mining methods by papers.

Technique	Classification	2010	2011	2012	2014	2015	2016	2017	Total(%)
TF-IDF	Term		#2	#5	#10	#12, #13, #15	#17	#26, #27	33.33%
Bigrams	Character			#3	#10		#20		11.11%
Unigrams	Character			#3			#20		7.41%
TF	Term		#2	#4					7.41%
Topic model	Concept						#20		3.70%
BM25ext	Term						#20		3.70%
BM25	Term			#3					3.70%
Binary Representation	Term		#2						3.70%

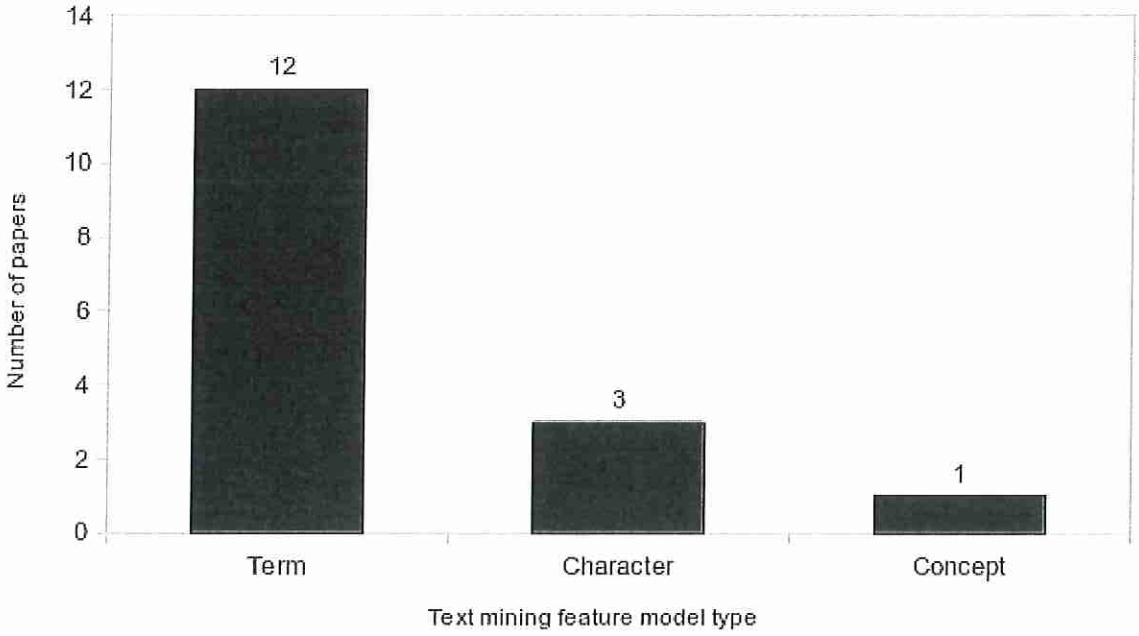


Fig. 5: Papers by text mining feature type

TABLE XII: ML algorithms distribution by papers.

Technique	Category	2010	2011	2012	2014	2015	2016	2017	Total(%)
KNN	Distance		#2	#3	#6, #8, #9	#12, #15	#18, #19, #20, #22	#26	44.44%
NB	Probabilistic	#1	#2	#5	#6, #8, #9, #10	#11, #12	#17, #22	#26	44.44%
NBM	Probabilistic		#2	#4	#7	#12, #13, #15	#19, #21, #23, #24	#25	40.74%
SVM	Optimization		#2	#5	#9		#19, #22	#26, #27	25.93%
Random Forest	Searching			#5	#8	#14	#17, #22		18.52%
C4.5	Searching			#5	#8, #16				11.11%
Bagging Ensemble	Ensemble					#14, #16			7.41%
Decision Tree	Searching						#19, #22		7.41%
AdaBoost	Ensemble						#17		3.70%
RBF Networks	Optimization						#17		3.70%
RIPPER	Other			#5					3.70%
Zero-R	Other				#8				3.70%

H. What are the measures used to evaluate ML algorithms performance for bug report severity prediction (RQ7)?

Table XII shows evaluations measures used in selected papers for bug report severity prediction. It can be observed that the most used evaluation measure for bug report severity prediction was precision (19 out of 27 - around 70%).

The chart in Figure 7 shows papers distribution by ML performance evaluation measure types. It can be observed that the most evaluate measure type used in bug report severity prediction was single class (21 out of 27 - around 70%).

I. Which sampling techniques are most applied to generate more reliable predictive performance estimates in severity prediction of a bug report (RQ8)?

Table XIV shows sampling methods used in selected papers for bug report severity prediction. It can be observed that sampling method preferred by authors of these papers was 10-Fold CV (10 out of 27 - around 37%).

The chart in Figure 8 shows papers distribution by sampling method type. It can be observed that the most sampling method type used in bug report severity prediction was simple re-sampling (15 out of 27 - around 55%).

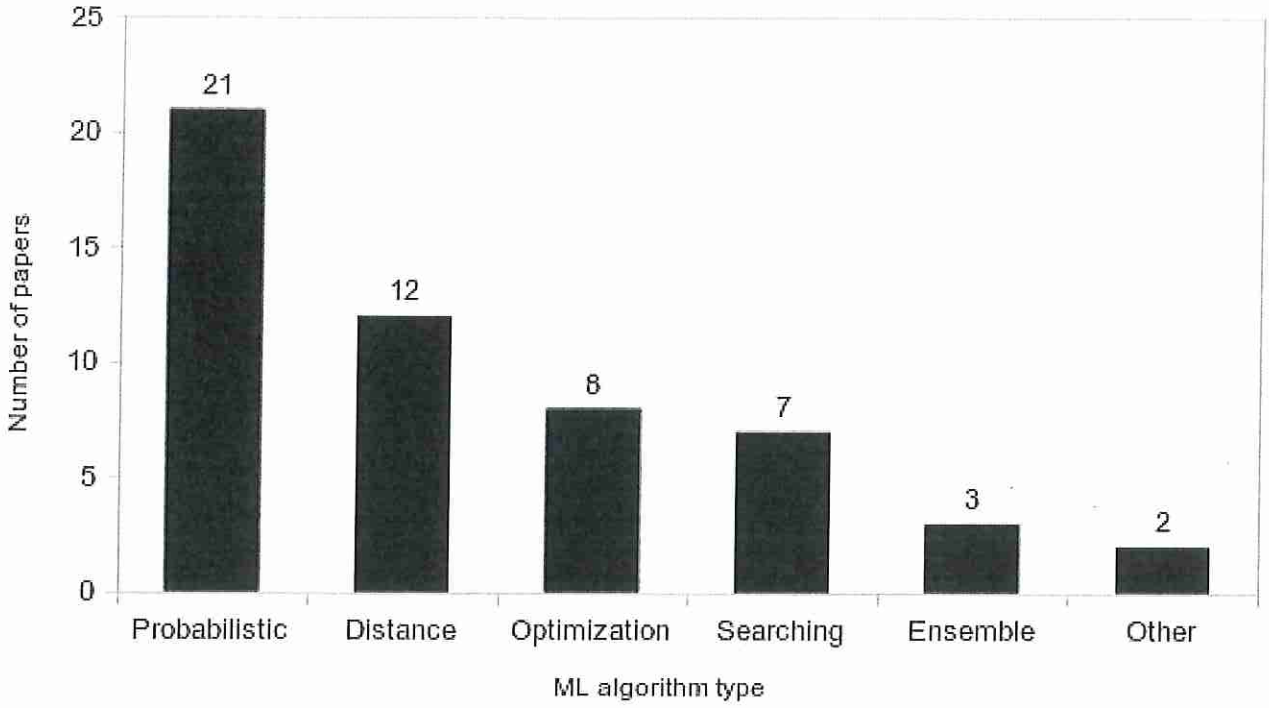


Fig. 6: Papers by ML algorithm categories

TABLE XIII: Evaluation measures used in selected papers.

Measure	Type	2010	2011	2012	2014	2015	2016	2017	Total(%)
Precision	Single class	#1		#3, #5	#6, #8, #9, #10	#12, #13, #14, #15	#16, #18, #20, #22, #24	#25, #26, #27	70.37%
F-measure	Single class	#1		#3, #5	#6, #8, #9	#12, #14, #10	#18, #20, #21, #22, #23, #24	#25, #26, #27	66.67%
Recall	Single class	#1		#3, #5, #12	#6, #8, #9, #10	#14	#16, #18, #20, #22, #24	#25, #26, #27	62.96%
Accuracy	Multi-class			#5	#6, #8, #9, #10	#11, #13, #15	#16, #17	#26	40.74%
AUC	Summary	#1	#2	#4	#7, #10				18.52%
ROC	Graphical	#1			#10				9.52%
MRR	Other						#6, #20		7.41%
Effectiveness Ratio@20%	Other					#14			3.70%
Krippendorff's Alpha Reliability	Other						#19		3.70%

TABLE XIV: Sampling methods used in selected papers.

Sampling Method	Type	2010	2011	2012	2014	2015	2016	2017	Total(%)
10-Fold CV	Simple re-sampling		#2	#4, #5	#7	#14	#16, #17, #23	#25, #27	37.04%
Stratified 10-Fold CV	Simple re-sampling				#8, #10			#26	11.11%
Hold-out	No re-sampling						#17		3.70%
SMOTE	Other						#22		3.70%
03-Fold CV	Simple re-sampling						#22		3.70%
05-Fold CV	Simple re-sampling					#13			3.70%

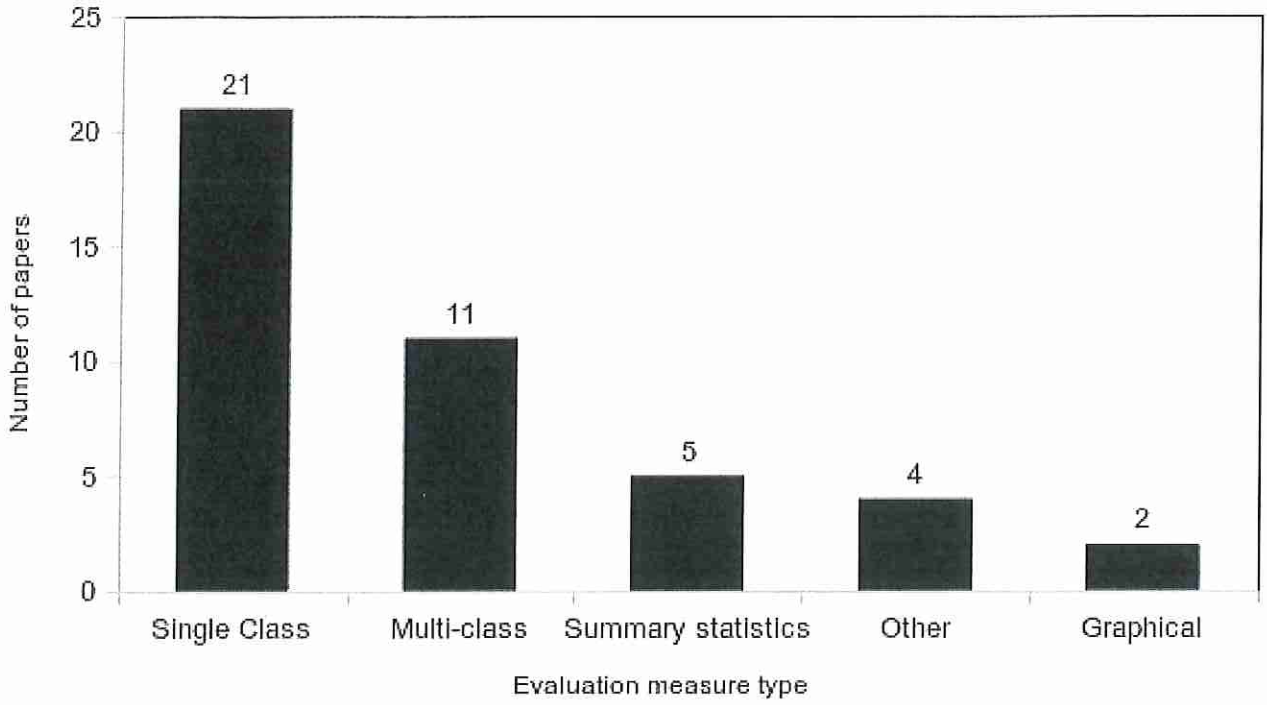


Fig. 7: Papers by evaluate measure types.

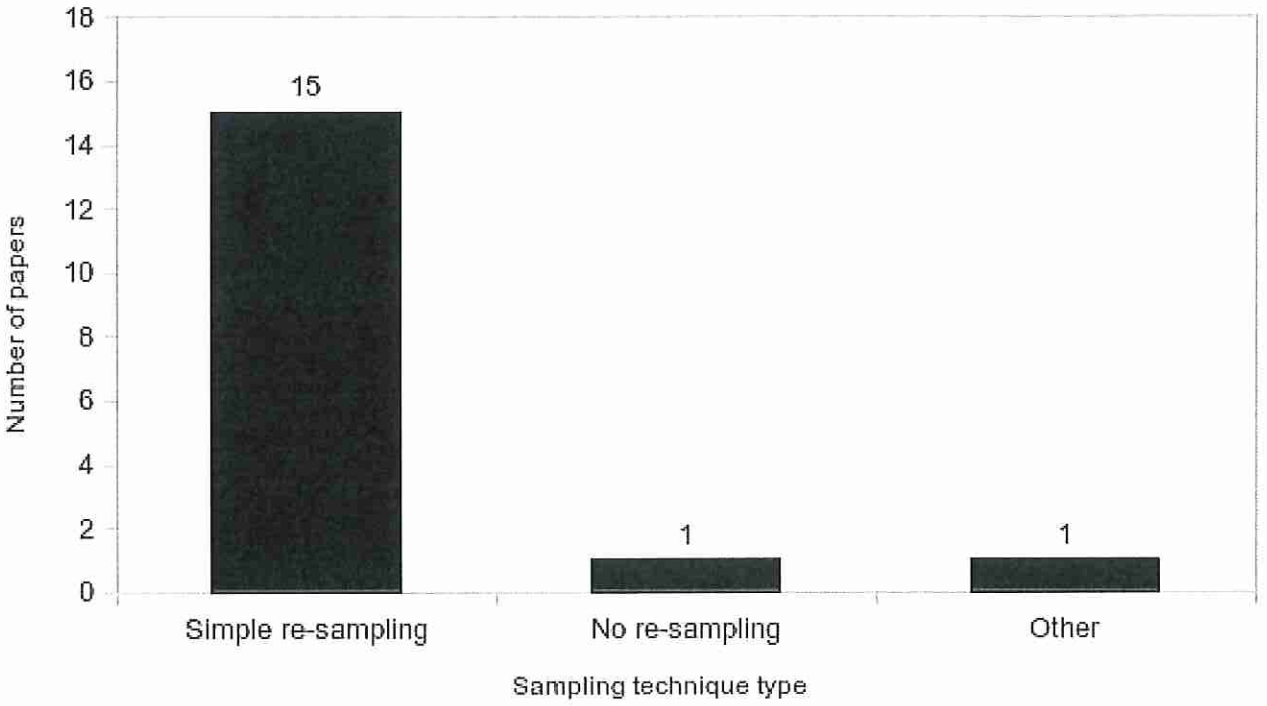


Fig. 8: Papers distribution by sampling method types.

TABLE XVI: Software tools used in selected papers.

Software	Type	2010	2011	2012	2014	2015	2016	2017	Total(%)
RapidMiner	CSS			#5	#9	#13, #15		#26	18.52%
WEKA	FLOSS		#2		#14, #16		#19		14.81%
R	FLOSS				#6	#12		#25	11.11%
NLTK	FLOSS				#10		#20, #23		11.11%
TMT	FLOSS				#6		#20		7.41%
Statisca	CSS			#5	#9				7.41%
Ruby	FLOSS	#1							3.70%
OpenNLP	CSS			#3					3.70%
WVTool	FLOSS			#4					3.70%
WordNet	CSS				#10				3.70%
CoreNLP	FLOSS						#25		3.70%

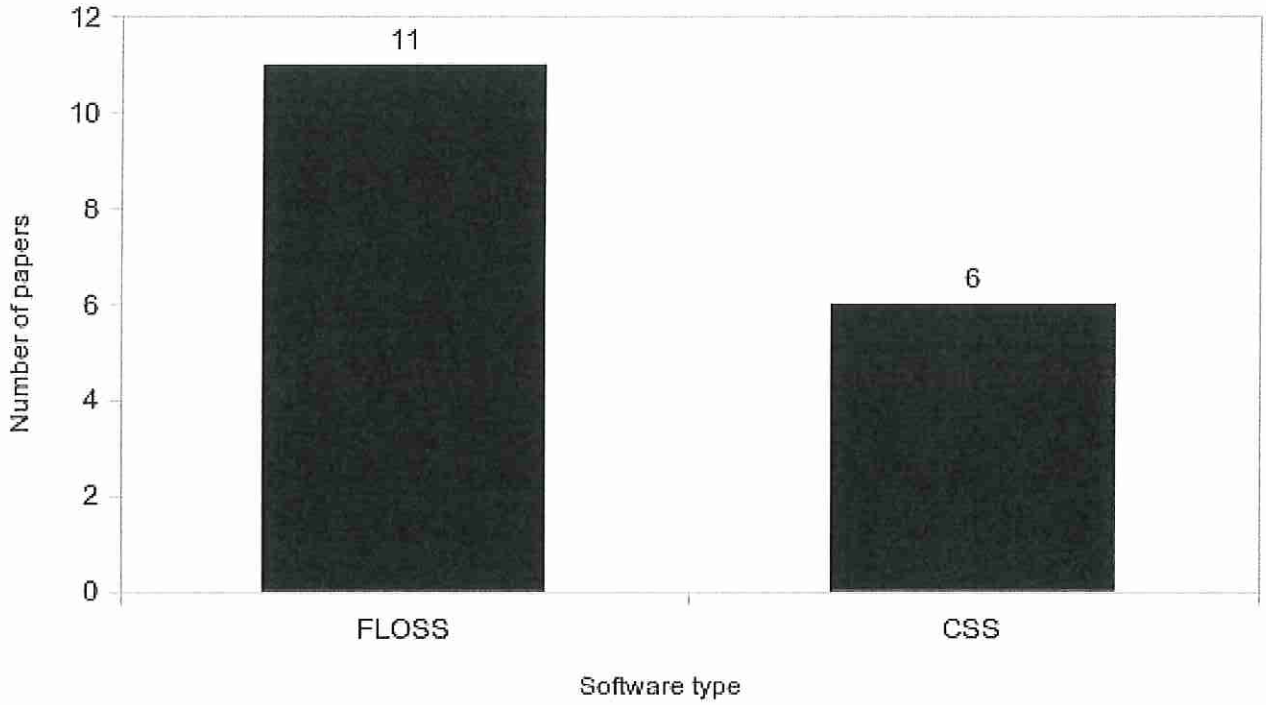


Fig. 10: Papers distribution by software tool types.

TABLE XVII: Solutions types proposed in selected papers.

Solution	2010	2011	2012	2014	2015	2016	2017	Total(%)
off-line	#1	#2	#4, #5	#6, #7, #8, #9, #10	#11, #12, #13, #14, #15	#16, #17, #18, #19, #20, #21, #22, #23, #24	#25, #26, #27	96.30
on-line			#3					3.70

J. What statistical tests was used to compare the performance between two or more ML algorithms for bug report severity prediction (RQ10)?

Table XV shows statistical tests used in selected papers for bug report severity prediction. It can be observed that the statistical tests most used by authors of these papers were T-test and Wilcoxon signed rank test (7 out of 27 - around 25%).

TABLE XV: Statistical tests used in selected papers.

Statistical Test	Type	2010	2011	2012	2014	2015	2016	2017	Total(%)
T-test	Parametric				#6	#12	#20, #22, #23	#25, #27	25.93%
Wilcoxon signed rank test	Parametric and non-parametric				#6, #8	#12, #14	#20, #23	#25	25.93%
Mann-Whitney U test	Non-parametric							#26	3.70%
Proportion test	Parametric							#27	3.70%
Shapiro-Wilk test	Parametric							#25	3.70%

The chart in Figure 9 shows papers distribution by statistical test type. It can be observed that the most statistical test type used in bug report severity prediction was parametric (9 out of 27 - around 33%).

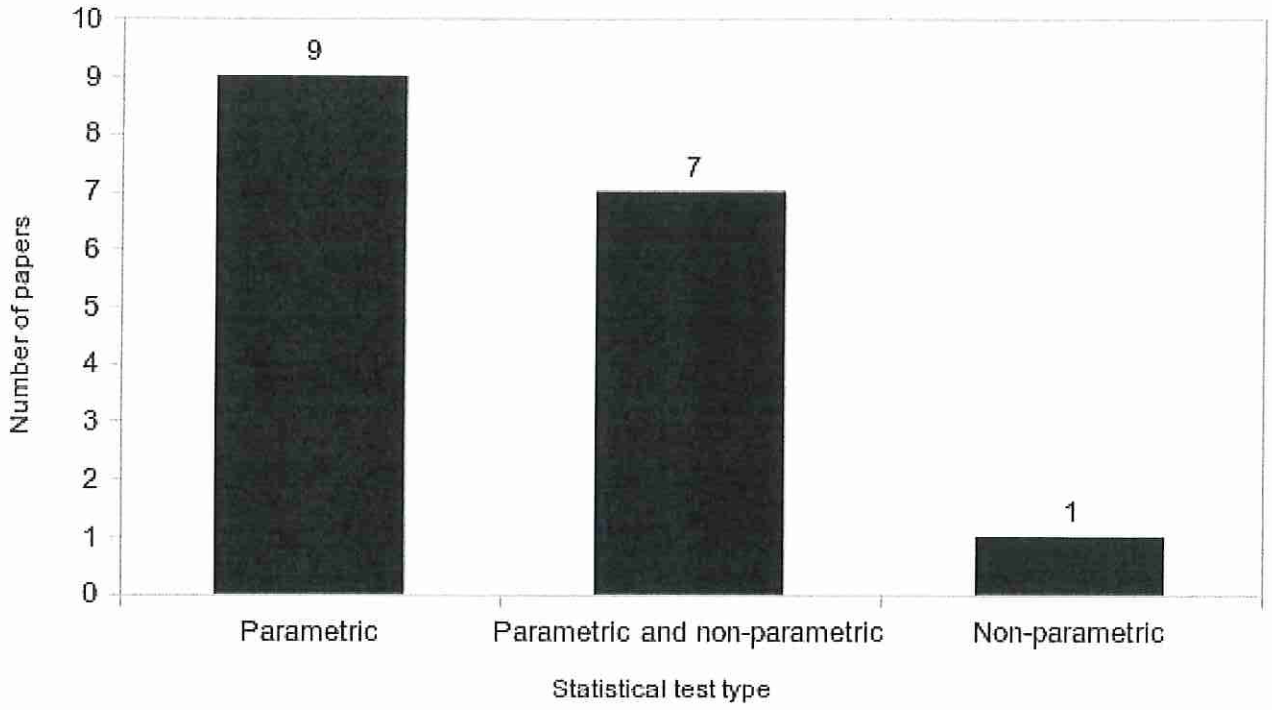


Fig. 9: Papers distribution by statistical test types.

K. What software tools were used to run bug report severity prediction experiments (RQ11)?

The Table ?? shows software tools used in selected papers for bug report severity prediction. It can be observed that the software tool most used by authors of these papers was RapidMiner (5 out of 18 - around 18%).

The chart in Figure 10 shows software tools distribution by select papers. It can be observed that the most software tools type used for bug report severity prediction was parametric (11 out of 27 - around 40%).

L. What solution types were proposed for bug report severity prediction problem (RQ12)?

Table XVII shows the solution types proposed in selected papers for bug report severity prediction. It can be observed that the most solution type was off-line (26 out of 27 - around 93%).

V. Discussion

This section discusses the outcomes which have been found and recommends possible improvements in this research area

The select papers used a total of 14 FLOSS projects in experiments for bug report severity prediction. Despite the number of these projects, one can notice the absence of remarkable FLOSS projects, mainly Linux Kernel and popular Linux distributions (e.g., Fedora) as bug reports sources. The authors weren't present any reason for this absence in the selected for this.

It's worth to observe that the vast majority of papers extracted and processed reports from two common projects: 24 out of 27 from Eclipse (corresponding to 88% of papers) and 19 out of 27 from Mozilla (corresponding 70% of papers). The main reason pointed by most of the authors[] to focus on these two project is to compare their solutions with the seminal papers of Lamkanfi [?], [10], who was first to establish the state-of-the-art in this researching area.

There was a definite focus on bug reports extracted from Bugzilla in the selected papers too, which was strongly influenced by choosing of projects. Consequently, these papers slightly investigated bug reports from Jira[], which tracks bugs for more than 80% Apache Foundation's projects, and Google Issue Tracker[], which tracks bugs for many internal and external Google projects. Also, It is worth mentioning that the selected papers did not in any way bugs reports from others prominent issue trackers, like one used by GitHub, an outstanding collaborative website, which stores more than 67 million of projects (including many FLOSS) [5].

Considering the software type of each FLOSS, six are programming tools, four are system software, and four are application software. Typically, the bug reporters of two first are mostly technical users and of the latter are end-users. A bug report can be more or less detailed depends on the reporter [?], [10]. It is expected a bug report written by technical users to be quite accurate and "good" than those written by end-users. Considering that 23 out of 27 FLOSS are programming tools or system software, it can conclude that technical users wrote the great majority of reports used for bug report severity prediction.

Summary and Description were the most used features for bug report severity prediction. They were utilized respectively in 81.5% and 63.0% of selected papers. Product and Component were other two features reasonably used in these papers (33% of them). These results suggest that proposed solutions are heavily based on unstructured textual information, and that bug reports from specific parts of the same FLOSS may generate different ML algorithms outcomes.

Regarding data type of available features in bug reports, more than 50% (14 out of 26) of them are qualitative. Essential ML algorithms like as Neural Networks and Support Vector Machines do not work with qualitative data [?]. These combined characteristics bring an extra difficult to use them for bug reports severity prediction. So, when an input data set has qualitative data, the values of these features should be converted into numeric values before to run these ML algorithms.

VI. Conclusions

i)Familiarize terms, objects, or processes.ii)indicate main purpose. iii) describe methods and materials.iv) describe results and explain them. v) outline research limitations. vi) outline research implications. vii) outline suggestions (future works). viii) cite authors' previous research. ix) cite previous research. x) discuss research contribution/importance

Acknowledgment

(omitted for double-blind reviewing).

References

[1] Y. C. Cavalcanti, P. A. da Mota Silveira Neto, I. d. C. Machado, T. F. Vale, E. S. de Almeida, and S. R. d. L. Meira, "Challenges and opportunities for software change request repositories: a systematic mapping study," *Journal of Software: Evolution and Process*, vol. 26, no. 7, pp. 620–653, jul 2014.

[2] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," 2007.

[3] M. Brhel, H. Meth, A. Maedche, and K. Werder, "Exploring principles of user-centered agile software development: A literature review," *Information and Software Technology*, vol. 61, pp. 163 – 181, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584915000129

[4] Érica Ferreira de Souza, R. de Almeida Falbo, and N. L. Vijaykumar, "Knowledge management initiatives in software testing: A mapping study," *Information and Software Technology*, vol. 57, pp. 378 – 391, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584914001335

[5] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE'08. Swindon, UK: BCS Learning & Development Ltd., 2008, pp. 68–77. [Online]. Available: http://dl.acm.org/citation.cfm?id=2227115.2227123

[6] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, Mar. 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=944919.944968

[7] R. Feldman and J. Sanger, *Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. New York, NY, USA: Cambridge University Press, 2006.

[8] K. Facelli, A. C. Lorena, J. Gama, and A. C. d. Carvalho, *Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina*. Rio de Janeiro, RJ, Brasil: LTC, 2015.

[9] N. Japkowicz and M. Shah, *Evaluating Learning Algorithms: A Classification Perspective*. New York, NY, USA: Cambridge University Press, 2011.

[10] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," *Proceedings - International Conference on Software Engineering*, pp. 1–10, 2010.

[5] https://octoverse.github.com/

VII. IDEIAS E DÚVIDAS

1) IDEIAS:

- Quais os principais problemas apontados na predição de severidade?
- padronizar as referências bibliográficas.
- de que maneira as publicações comparam o desempenho do ser humano com o problema?
- substituir a classificação de software type por reporter kind (end user, software developer, automatic reporter)
- substituir a classificação de features types(kind) por (categorical, ordinal, quantitative)

2) DÚVIDAS:

- Mostrar algoritmos de ML criados ou adaptados no artigos ou mostrar o algoritimo que originou (mais popular) ?
- o que fazer quando um item não possui uma classificação?
- Estudar relief feature method.
-