Artif Intell Rev
DOI 10.1007/s10462-016-9478-6

A survey on bug prioritization

Jamal Uddin[1] · Rozaida Ghazali[1] ·
Mustafa Mat Deris[1] · Rashid Naseem[1] · Habib Shah[2]

Abstract Daily large number of bug reports are received in large open and close source bug tracking systems. Dealing with these reports manually utilizes time and resources which leads to delaying the resolution of important bugs. As an important process in software maintenance, bug triaging process carefully analyze these bug reports to determine, for example, whether the bugs are duplicate or unique, important or unimportant, and who will resolve them. Assigning bug reports based on their priority or importance may play an important role in enhancing the bug triaging process. The accurate and timely prioritization and hence resolution of these bug reports not only improves the quality of software maintenance task but also provides the basis to keep particular software alive. In the past decade, various studies have been conducted to prioritize bug reports using data mining techniques like classification, information retrieval and clustering that can overcome incorrect prioritization. Due to their popularity and importance, we survey the automated bug prioritization processes in a systematic way. In particular, this paper gives a small theoretical study for bug reports to motivate the necessity for work on bug prioritization. The existing work on bug prioritization and some possible problems in working with bug prioritization are summarized.

Keywords Survey · Bug report · Bug prioritization · Bug triaging · Classification · Clustering

1 Introduction

In large software development systems, bug triaging is an important activity of software testing. It helps in software bug management, while taking critical decisions related to the

✉ Jamal Uddin
jamal_maths@yahoo.co.uk

1 Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Batu Pahat, Malaysia

2 Faculty of Computer and Information Systems, Islamic University of Madinah, Madinah, KSA

software bugs fixing. It involves process like assigning appropriate developer to the bug who could fix it, analyzing which bug needs immediate attention and which one does not, and finding duplicated ones. However, manual bug triaging becomes quite time consuming and tedious because being a significant part of software development it takes up considerable amount of time and resources. As of August 2009, the Mozilla bug database contains over 500,000 and the Eclipse bug database over 250,000 bug reports. On average, Mozilla received 170 and Eclipse 120 new bug reports on each day from January to July 2009 (Guo et al. 2010). So, we can say that in case of large and open source software systems, large number of bugs are reported per day, which ultimately increases the software maintenance time and cost.

In the process of bug triage, a triager takes a decision about the bugs entered in the bug repository by analyzing them in two ways. In repository-oriented decisions, after verifies that the reported bug is not a duplicate bug, the triager checked it for validity, i.e. is it a real bug or not. The motivation behind these decisions is to remove the bug reports that do not need to be resolved. Remaining bug reports are investigated for development-oriented decisions where the triager examine severity and priority levels of the bugs. These levels are changed by him if inappropriate, so that critical bugs will be given time and resources (Anvik and Murphy 2011). After this, triager writes remarks for the bug and assigns this bug report to the suitable developer to resolve the bug. Hence there arises a need to develop an automated system for bug triaging so as to save developer's time and efforts, ultimately leading to save resources of the organization. Researchers addressed this problem by automating its various aspects like identification of duplicate bugs in Jalbert and Weimer (2008) and Lazar et al. (2014). Assigning a bug to an appropriate developer who could fix it in Cubranic (2004) and Zhang and Lee (2013). Kremenek and Engler (2003), Jaweria Kanwal (2010) and Yang et al. (2014) worked on classifying the priority of bugs. Similarly, severity of the bugs in Jianhong et al. (2010), Chaturvedi and Singh (2012) and task of grouping similar bug reports was performed in Anvik et al. (2005), Nagwani and Verma (2012).

As an essential part of bug triaging process the bug prioritization task becomes very important for big size projects specially an open source project because the responsiveness of a project is usually measured by the number of outstanding bug reports in the repository and how quickly a bug report is addressed (Jaweria Kanwal 2010). Bug prioritization process usually performed manually which makes it error-prone and require intensive effort. It depends massively on the triager perception and experience. Many bug reports may have been assigned incorrect priority levels and many of them may usually left blank. Wrong assignments of priority levels may result ineffective utilization of resources (for example time and efforts need to fix unimportant bugs first) (Alenezi and Banitaan 2013). To prioritize bug reports developers need an automated approach to solve the aforementioned problems. For achieving these tasks, researchers have used several machine learning approaches such as Decision Tree (DT) by Giger et al. (2010), Alenezi and Banitaan (2013), Garcia and Shihab (2014), Support vector machine(SVM) classification algorithm by Jaweria Kanwal (2010), Chaturvedi and Singh (2012), Naive Bayes (NB) classifier by Lamkanfi et al. (2010), Chaturvedi and Singh (2012) and Random Forest (RF) by Alenezi and Banitaan (2013), Goyal et al. (2015). Similarly, researchers have published surveys in this bug triaging field like Punitha and Chitra (2013) and Zhang et al. (2015), where they have investigated bug report analysis and software defects prediction using software metrics. Despite of reasonable work published on assigning appropriate priority level to the reported bug, our research study particularly focuses on a survey of bug prioritization. The main objectives of this research can be summarize as follows,

1. To review the existing literature for exploring what and how much work is published so far.
2. To track the trends of research in this area,
3. To identify the significances of this field,
4. To identify the problems faced by researchers and
5. To present the impacts of automated bug prioritization

Kitchenham and Charters (2007) systematic literature survey approach has been adopted for this survey to reduce the possibility of researcher bias. This paper reports our study plan in the form of two survey protocols from year 2003 to 2015 using five online search engines and the selected search terms. The rest of this paper is organized as follows. Section 2 presents some terms mostly used in bug prioritization and bug triaging. Section 3 shows a comparative study of existing surveys and our survey. Section 4 explains our research method. Section 5 reviews the result of our survey by exploring existing work on bug prioritization. Section 6 presents significance outcomes of this survey. Section 7 summarize some research gaps that were explored during the survey, and finally, Sect. 8 concludes the paper.

2 Preliminary

This section, explains some basic concepts of bug prioritization method, life-cycle of bug reports, useful attributes as well as terms related to bug triage system.

2.1 Software bug, bug reporting and bug tracking system

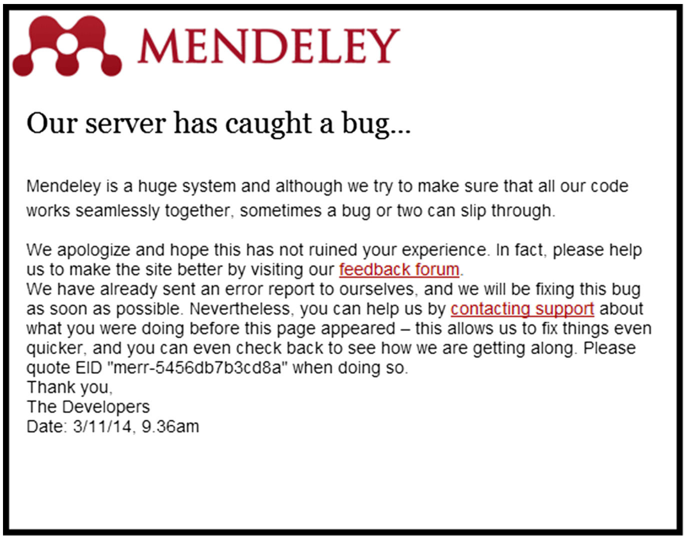
A defect or a flaw in the software is called a *bug*, it indicates the unexpected behavior of a software system (Nagwani and Verma 2011). Mostly, bugs arise from mistakes made by the development team in gathering requirements, designing or in coding and a few are caused by compilers. A program with large number of bugs is known as *buggy program* (Nigam et al. 2012). A bug report is a document that is submitted by a developer, tester, or end-user of a system. It describes the defect(s) that occurred. Such documents generally describe the situations in which the software does not behave as it is expected, i.e. fails to fulfill the expected requirements.

This bug reporting and fixing is an important phase of software development, refinement and maintenance for both open and close source projects. Various bug reporting systems have been developed for submission or reporting of a bug and tracking their progress of fixing. Bug reporting and tracking systems provide a platform to record the problems/failures faced by the client or user of the software. Nowadays, users of software systems are encouraged to report the bugs they encounter, using bug tracking systems such as Jira[1] and Bugzilla.[2] These systems are designed for quality assurance and helps the programmers to keep track of reported software bugs in their work. These reporting systems allow users to report, track, describe, comment on and classify bug reports and feature requests. An example of bug appear and detected in Mendeley[3] software is depicted in Fig. 1. While, Fig. 2 presents the overview and content detail of a sample bug report taken from Eclipse bug repository.

1 www.atlassian.com/software/jira.

2 www.bugzilla.org.

3 www.mendeley.com.



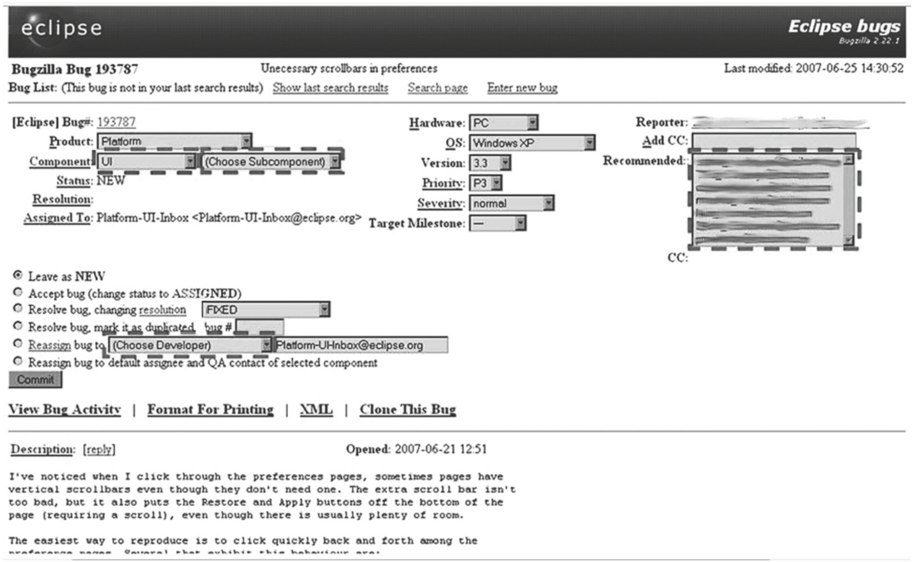
MENDELEY

Our server has caught a bug...

Mendeley is a huge system and although we try to make sure that all our code works seamlessly together, sometimes a bug or two can slip through.

We apologize and hope this has not ruined your experience. In fact, please help us to make the site better by visiting our feedback forum.
We have already sent an error report to ourselves, and we will be fixing this bug as soon as possible. Nevertheless, you can help us by contacting support about what you were doing before this page appeared – this allows us to fix things even quicker, and you can even check back to see how we are getting along. Please quote EID "merr-5456db7b3cd8a" when doing so.
Thank you,
The Developers
Date: 3/11/14, 9.36am

Fig. 1 Bug caught in Mendeley



Bugzilla Bug 193787 Unecessary scrollbars in preferences Last modified: 2007-06-25 14:30:52

Bug List: (This bug is not in your last search results) Show last search results Search page Enter new bug

[Eclipse] Bug#:	193787	Hardware:	PC	Reporter:	[redacted]
Product:	Platform	OS:	Windows XP	Add CC:	
Component:	UI / (Choose Subcomponent)	Version:	3.3	Recommended:	[redacted]
Status:	NEW	Priority:	P3		
Resolution:		Severity:	normal		
Assigned To:	Platform-UI-Inbox <Platform-UI-Inbox@eclipse.org>	Target Milestone:	---	CC:	

[✓ Leave as NEW]
Accept bug (change status to ASSIGNED)
Resolve bug, changing resolution FIXED
Resolve bug, mark it as duplicated bug #
Reassign bug to (Choose Developer) Platform-UI-Inbox@eclipse.org
Reassign bug to default assignee and QA contact of selected component

Commit
View Bug Activity | Format For Printing | XML | Clone This Bug

Description: [reply] Opened: 2007-06-21 12:51

```
I've noticed when I click through the preferences pages, sometimes pages have
vertical scrollbars even though they don't need one. The extra scroll bar isn't
too bad, but it also puts the Restore and Apply buttons off the bottom of the
page (requiring a scroll), even though there is usually plenty of room.

The easiest way to reproduce is to click quickly back and forth among the
```

Fig. 2 Bug report sample from Anvik and Murphy (2011)

2.2 Bug repository

A bug repository is one of the most important software repositories, which is considered as a vital data base in modern software developments. Both users and developers can report bugs to bug repositories supported by software development projects. These projects are burdened by the rate at which new bug reports appear in the bug repository. One potential advantage of an open bug repository is that it may allow more bugs to be identified and solved, improving the quality of the software.

The widely available bug repositories have provided an important platform for investigating the quality of software. With the growth in scale, developers in large projects must handle a large number of bugs in bug repositories. For example, from Oct. 2001 to Dec. 2010, the bug repository of an open source project, Eclipse has recorded 333371 bugs (Xuan et al. 2012). There are at least two important advantages of such type of a bug repository. First, the bug repository allows users all around the world to be testers of the software, so it can increase the possibility of revealing defects and thus increase the quality of the software. Second, it helps the software evolve according to users requests, and meet the requirements of more users.

2.3 Bug attributes

A software bug reports have many attributes, some of which are filled at the time of reporting and others are filled during the process of fixing. Some attributes are qualitative in nature but some are quantitative. A clear understanding of bug attributes, their interdependence and their contribution in predicting the other attributes will help in improving the quality of software. Table 1 presents an example of bug repository which shows bug report features and class taken from Eclipse bugs data set. Some attributes are categorical such as bug-id, date of submission, component, product, resolution, status, severity (how serious bug is), priority (how important bug is, represented normally as levels P1-P5 with P1 being most important), platform, operating system, reporter, assignee, cc-list, and some are text fields such as summary and long description. Some of the categorical fields are fixed at the time of report submission, e.g. bug id, report submission time and reporter name. Some fields such as product, component, severity, priority, version, platform and operating system are entered by the reporter but may be changed by the triager or developer if needed (Jaweria Kanwal 2010). Bug attributes also comprises of developer who resolves the bug, list of people who are interested in bug resolution, bug-status, and resolution, change throughout bug life time.

The free form text includes the title of report, a full description of the bug, and additional comments. The full description typically contains an elaborated description of the effects of the bug and any necessary information for a developer to handle bug report. The additional comments include discussions about possible approaches to fixing the bug, and pointers to other bugs that contain additional information about the problem or that appear to be duplicate reports. Reporters and developers may provide attachments to reports to provide non-textual additional information, such as a screen shot of erroneous behavior (Anvik et al. 2006).

2.4 Interactions with bug report

People play different roles as they interact with reports in a bug repository. The person who submits the report is known as *reporter* or the submitter of the report. The *triager* is the person who decides if the report is meaningful and who assigns responsibility of the report to a developer. The one that resolves the report is the *resolver*. A person that contributes a fix for a bug is called a contributor. A contributor may also contribute comments about how to resolve a bug or additional information that leads to the resolution of a report. A person may assume any one of these roles at any time. For example, a triager may resolve a report as the duplicate of an existing report. Alternatively, a developer may submit a report, assign it to himself, contribute a fix, and then resolve the report. For that report, a single person has fulfilled all the roles (Anvik et al. 2006).

Table 1 Bug repository: bug report features and class label

Bug ID	Product	Component	Summary	Opened	OS	Class
9649	JDT	Debug	Can't rebuild all, never returns	2/13/2002 11:01	Linux-Motif	P1
80406	Platform	UI	[DynamicUI] NPE trying to open GIF file	12/7/2004 14:24	Windows XP	P1
6582	JDT	Debug	VM Preference Page, Checkbox Table Viewer has no columns anymore	12/5/2001 7:17	All	P1
40673	JDT	Debug	Detected JRE not set correctly	7/23/2003 13:50	Linux	P1
77462	Platform	Debug	CVS Plug-in fails to start	11/2/2004 6:40	Windows XP	P1

2.5 Bug triaging system

According to Hooimeijer and Weimer (2007), triage is the act of inspecting a bug report, understanding its contents, and making the initial decision regarding how to address the report. In the normal flow of the bug process, someone discovers a bug and creates the respective bug report, then the bug is assigned to a developer who is responsible for fixing it and finally, once it is resolved, another developer verifies the fix and closes the bug report. Bug triaging is analyzing these bug reports to determine, for example, duplicate or unique, important or unimportant, and who will resolve them. It is one of the important software maintenance activities. The task of triaging becomes very important for software projects because the responsiveness of a project is usually measured by the number of outstanding bug reports in the repository and how quickly a bug report is addressed (Jaweria Kanwal 2010).

2.6 Life cycle of bug fixing

In IEEE Standard 1219 (Bennett and Rajlich 1999), the software maintenance is defined as, "The modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment". Among the types of software maintenance, the corrective and adaptive get the most attention (Chapin 2000). During corrective maintenance, the programmer has to understand the software enough to analyze a problem, locate the bug and determine how it should best be fixed without breaking anything (VANS 1999). There are different states in which a bug report can experience in its life-cycle. Figure 3 depicts the life-cycle of bugs in Bugzilla-based projects. When a new bug report is filed, it is assigned a NEW state. Once it has been triaged and assigned to a developer, its state is then changed to ASSIGNED. After closing this bug, its state is set to RESOLVED, VERIFIED or CLOSED. The resolution to this bug is marked in several ways; the resolution status in the report is used to record how the report was resolved. If the resolution results in changing code base, this bug is marked as FIXED. When a bug is considered as a duplicate to other bugs, it is set to DUPLICATE. If a bug will not be fixed,

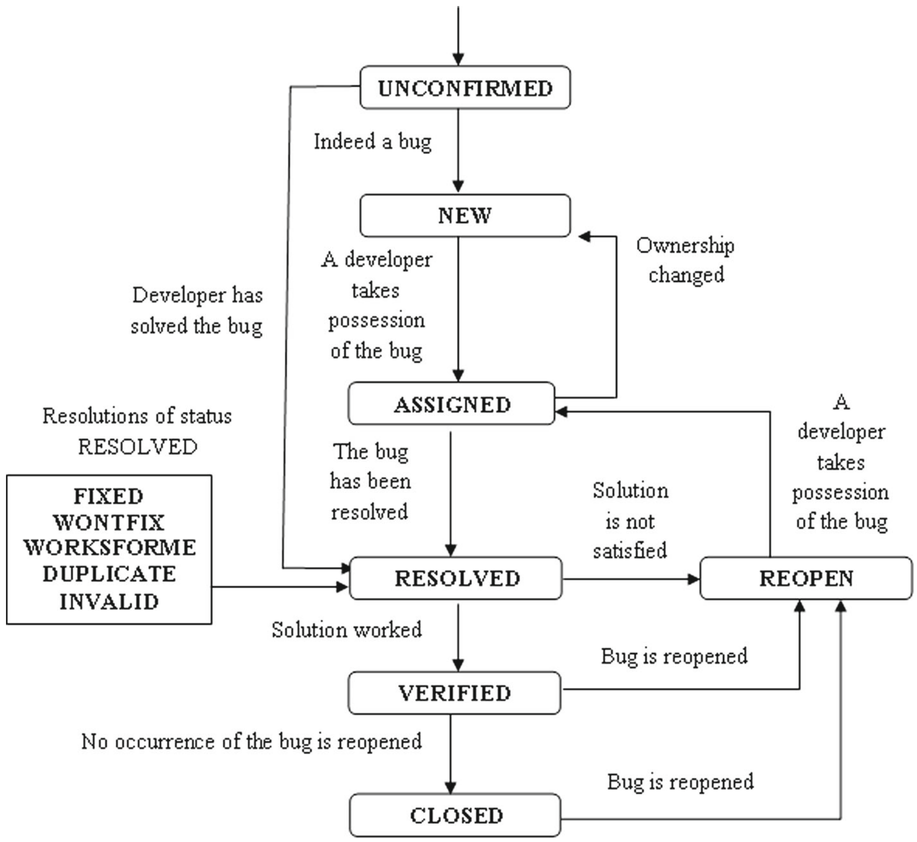

UNCONFIRMED
Indeed a bug
NEW
Ownership changed
A developer takes possession of the bug
Developer has solved the bug
ASSIGNED
Resolutions of status RESOLVED
FIXED
WONTFIX
WORKSFORME
DUPLICATE
INVALID
The bug has been resolved
Solution is not satisfied
A developer takes possession of the bug
RESOLVED
REOPEN
Solution worked
Bug is reopened
VERIFIED
No occurrence of the bug is reopened
Bug is reopened
CLOSED

Fig. 3 Bug report life cycle (Zhang et al. 2015)

or it is not an actual bug, it will be set to WONTFIX or INVALID respectively. If a bug was resolved but has been reopened, it is marked as REOPENED (Alenezi and Banitaan 2013).

2.7 Why bug prioritization?

Software developers spend a significant portion of their resources handling user-submitted bug reports. For a software that is widely deployed, the number of bug reports typically outstrips the resources available to triage them. As a result, some bug reports may be dealt with too slowly or not at all. Boehmand Basili claims that maintenance consumes over 70 % of the total life cycle cost of a software product (Hooimeijer and Weimer 2007). In Anvik et al. (2006) article on page 361, it is mentioned that “Consider the case of the Eclipse open source project over a four month period (January 1, 2005 to April 30, 2005) when 3426 reports were filed, averaging 29 reports per day. Assuming that a triager takes approximately five minutes to read and handle each report, two person-hours per day is to be spent on this activity. If all of these bugs led to improvements in the code, this might be an acceptable cost to the project”. Main aim of automated bug prioritization is reducing the time spent triaging.

Bug tracking systems allow users to select a severity for the bug they have found. Users assign severity, and developers give priority to the reports depending on their severity. One of the major problems when assigning severity to a bug is that a reporter might not be the best qualified to determine exactly what type the defect falls into. Another issue is that to submit a report it is required to know the precise difference between the categories, and this is not always the case (Herraiz et al. 2008). Developers are usually unable to cope with all the bugs that are notified as they are submitted. They need to prioritize them. They need to determine, for any particular bug, how important it is, and allocate their time and resources according to such prioritization. Correct prioritization of bugs helps in bug fix scheduling/assignment and resource allocation. Failure of this will result in delay of resolving important bugs (Sharma et al. 2012).

2.8 Prioritization levels of bug

During bug triaging, a software development team decide how soon bugs needs to be fixed, using different categories. For example, researchers like Yu et al. (2010) and Lamkanfi et al. (2011) used four priority categories that are (P1) as soon as possible; (P2) before the next product release; (P3) may be postponed; (P4) bugs never to be fixed. Five category priority level scheme (that is P1, P2, P3, P4, and P5) adopted by most researchers for example Kanwal and Maqbool (2012) and Garcia and Shihab (2014). Also, in Bugzilla there are same 5 priority levels:P1 is the highest priority and often the software system can only be shipped if these high priority bugs are fixed. P5 on the other hand is the lowest priority and bugs assigned this priority might remain unfixed for a long period of time (Tian et al. 2013). If the bug triager is uncertain about the priority of a bug or it is actually a normal bug, she can set P3 priority which is the default priority (Saha et al. 2014). Then the assigned developer can adjust it if appropriate.

3 Related work

Researchers have been studying on bug report prioritization, detecting duplicate bugs and developer assignment. In addition, there are a number of studies compiling up-to-date developments in this field. In other words, some researchers focus on surveying about bug reports and their triaging.

Punitha and Chitra (2013) presented a survey on software defect prediction using software metrics. They focuses in identifying defective modules and the scope of software that needs to be examined for defects can be prioritized. The goal of their research is to help developers identify defects based on existing software metrics using data mining techniques and thereby improve software quality which ultimately leads to reducing the software development cost in the development and maintenance phase.

A survey on clustering techniques in data mining for software engineering was conducted by Kaur and Garg (2014). They explore the useful data mining techniques for software engineering tasks of programming, testing, bug detection, debugging and maintenance. In their survey they provide the discussion of data mining specifically clustering techniques. They concluded that every technique has to solve different problems and have their own advantages and disadvantages. There is no such clustering technique and algorithm exists that is used to solve all the problems and is a best fit for all applications.

Mishra and Kumar (2015) presented survey on types of bug reports and general classification techniques in data mining. Their study shows types of bug reports and various classification techniques widely used in data mining like statistical and soft computing approaches. Their work shows importance of classification techniques in data mining process especially when dealing with bug reports.

To review the work on bug-report analysis, Zhang et al. (2015) presented an exhaustive survey on the existing work on bug-report analysis. After introducing some preliminaries, they presented some statistics on practical bug reports to show that ensuring bug reports quality, automating bug reports triage and localization are indeed urgent to be solved. Then they conducted a rather thorough survey on existing bug-report analysis work, mainly including bug-report optimization, bug-report triage, and bug fixing. They concluded that among the work in bug-report analysis, machine learning and information retrieval are main techniques widely used. Also, researchers need to improve the accuracy of existing automatic approaches.

We summarize the above mentioned surveys and current survey with respect to contents of several aspects of bug prioritization in Table 2. The "generally analyzed" are the topics that are just defined or generally discussed. Whereas, "deeply analyzed" are those topics that are discussed with more detail in concerned articles. It can be seen from this table that our survey deeply analyzed the several contents of bug prioritization as compared to other surveys. Though, Zhang et al. (2015) tried to cover wide range of work on bug report thoroughly, but our survey will explore the field of bug prioritization specifically.

4 Method

We follow Kitchenham approach (Kitchenham and Charters 2007) for conducting this survey. Kitchenham defines three main steps for a systematic review process: planning the review, conducting the review, and reporting the review. He suggested that a pre-defined protocol is necessary to reduce the possibility of researcher bias. We followed the guidelines for conducting systemic literature review which have also been proposed in previous examples Kitchenham and Charters (2007) and Brereton et al. (2007).

The collected articles are published during the years 2000 to 2015. Moreover, a Bib TeX library using Mendeley is developed for managing the papers. In this survey, we reviewed 84 journal and conference papers. There was a two-step review process and protocol built for cutting down the paper number to 32. Both qualitative (Sect. 5) and quantitative (Sect. 6)analysis were conducted for the selected papers.

4.1 Review protocol Step 1 (84 papers)

For our survey, five search venues are targeted. They are:

1. Institute of Electrical and Electronics Engineers (IEEE Xplore)
2. Association for Computing Machinery (ACM Digital Library)
3. Science Direct
4. Wiley (Wiley Online Library)
5. The Institution of Engineering and Technology (IET)

The related papers which were not found in the given 5 venues were downloaded by using Google Scholar and are added in "Other" category. Our search terms help to determine the scope of our definition of bugs prioritization since many of the terms include the word "Bugs" such as; bugs classification, duplicate bugs, bug fixing, bug assignment, bug triaging, and bug tracking system. The search conditions and terms that we used included terms for bugs prioritization in conjunction with terms for possible outcomes, impacts or effects of bugs prioritization process. Some of search terms are chosen from frequent key words in the bug

Table 2 Comparison of previous surveys and current study

Comparison attributes	Survey papers				
	Zhang et al. (2015)	Mishra and Kumar (2015)	Kaur and Garg (2014)	Punitha and Chitra (2013)	Current survey
Bug report					
Preliminaries	**	*	*	*	**
Issues	**	*	*	*	**
Analysis	**	–	–	–	*
Bug prioritization approaches					
Classification	*	**	–	*	**
Information retrieval	*	*	–	–	**
Classification by clustering	*	–	**	–	**
Other techniques	*	*	–	–	**
Evaluation components					
Data sets	–	*	–	–	**
Metrics	–	–	–	**	**
Discussion	–	–	–	*	*
Significance of survey					
Systematic survey	*	–	–	–	**
Significance findings	**	–	*	–	**
Future directions	**	–	*	–	**

–, not mentioned; *, generally analyzed; **, deeply analyzed

prioritization articles while, remaining search terms are selected manually that covers the scope of bug prioritization. Search terms used for our survey are:

– Bug priority
– Bug prioritization
– Bug priority classification
– Reported bug priority prediction
– Bug report priority prediction

84 papers were found from academic journals, conference proceedings and book chapters worldwide. Table 3 summarizes the total retrieved and filtered results.

4.2 Review protocol Step 2 (32 papers)

Step two reduces the number of research papers from 84 to 32, by applying the selection criteria that is,

1. Removing the duplicate papers that is the papers that are retrieved by more than one search terms.

Table 3 Search terms retrieved and filtered results

Search terms	IEEE		ACM		SD		Wiley		IET	
	R	F	R	F	R	F	R	F	R	F
Bug priority	45	8	2659	20	544	0	3411	1	43	0
Bug prioritization	31	7	554	3	544	0	393	1	0	0
Bug priority classification	6	4	619	12	1487	1	1009	0	2	0
Reported bug priority prediction	7	6	272	12	767	0	746	0	8	0
Bug report priority prediction	7	5	358	4	948	0	746	0	8	0
Total	96	30	4462	51	4290	1	6305	2	61	0

R retrieved, *F* filtered, *SD* Science Direct

Table 4 Review protocol Step 1 and Step 2 summary

	Step 1 filtration	Step 2 filtration
IEEE	30	18
ACM	51	5
Science Direct	1	0
Wiley	2	0
IET	0	0
Others	0	9
Total	84	32

2. Selecting papers specifically dealing with bug prioritization process.
3. Removing papers that are dealing with other tasks of bug triaging system like developer assigning, duplicate bug detection and bug report analysis.

Table 4 summarizes the review protocol Step 1 and Step 2 results. While Table 5 gives year wise summary of final results. Based on this two-step review protocol, next section describes and analyzes the collected data from the 32 chosen reviewed papers.

5 Literature review discussion

In this Sect. 32 papers of the researchers that worked on bug prioritization are reviewed. Different data mining approaches like classification, information retrieval, classification by clustering and categorization were adopted to prioritize bug reports. Studies about bug prioritization using data mining techniques can be grouped into three major classes. The first group consists of studies focusing on main classification and information retrieval approaches. The second group of researchers focuses on categorization bug reports in order of importance. The last group of studies performs other novel approaches to deal with bug prioritization issue. Each group will be discussed separately in subsequent subsections.

5.1 Classification and information retrieval techniques

Podgurski et al. (2003) proposed automated support for classifying reported software failures so as to facilitate prioritizing and diagnosing their causes. A classification strategy

Table 5 Year wise summary of filtered papers

Year	No. of filtered papers
2003	2
2007	2
2008	1
2009	1
2010	4
2011	2
2012	10
2013	2
2014	7
2015	1
Total	32

that involves the use of supervised and unsupervised pattern classification and multivariate visualization is presented. The resulting classification is then used to assess the operational frequency and severity of failures caused by particular defects and to diagnose those defects. The methodology involved is subject programs inputs, feature selection, cluster analysis, visualization and manual examination. With a relative simple type of classifier and with coarse grained execution profiles the results proved it to be an effective strategy.

Artificial Neural Network(ANN) technique was used to predict defect priorities by Yu et al. (2010). They improved the efficiency of troubleshooting, by proposing to employ neural network technique to predict the priorities of defects, adopt evolutionary training. A framework is built up for the model evaluation, and a series of experiments on five different software products of an international healthcare company to demonstrate the feasibility and effectiveness. The threefold cross-test in both the closed test and opening test ways were executed. Compared with Bayes algorithm, the ANN model showed better qualification in terms of recall, precision and F-measure. The comparison was carried on the RIS2.0 software project with sample size about 2000 bug reports.

Meanwhile a classification-based approach to create a bug priority recommender was presented by Jaweria Kanwal (2010), which assigns a priority level to new bug reports in a bug repository. Priority assignment assists triager in resolving the important bugs first. The main contributions were proposing the machine learning based approach SVM for automatic assignment of bug priority to new bug reports in open source bug repository, exploring the bug report attributes that contribute more towards determining the priority of a bug and evaluating the affect of training data set size on the accuracy of bug priority recommender. Experimental evaluation of their recommender using precision and recall measures reveal the feasibility of their approach for automatic bug priority assignment.

Lamkanfi et al. (2010) in the first stage investigated whether they can accurately predict the severity of a reported bug by analyzing its textual description using text mining algorithm Naive Bayes classifier. They evaluated the performance of predictions based on three cases drawn from the open-source community. This study motivates researchers to implement a more automated and more efficient bug triaging process. They investigated whether the longer description included in a bug report would result in a better predictor. Performance of their approach stabilizes in terms of precision and recall for GNOME as compared to Mozilla and

Eclipse. They concluded that how soon a reported bug needs to be fixed partly depends on its severity.

In the second stage Lamkanfi et al. (2011), reported on a follow-up study where they compare four well-known text mining algorithms with respect to accuracy and training set size. They discovered that for the cases under investigation of two open source systems which algorithm performs superior compared to the other proposed algorithms. Steps involved in their approach are extract and organize and preprocessing the bug reports, training of the predictor, predicting the severity and hence concluding that Naive Bayes Multinomial is best suited for the purpose of classifying bug reports. They also deduce from the resulting classifiers, that the terms indicating the severity of a bug report are component dependent. Their work contributed well in a way that current research can be combined with this approach to improve the overall reliability of more automated bug triaging process.

To demonstrate the applicability of various machine learning algorithms in determining the class of bug severity based on the textual summary of the bug report Chaturvedi and Singh (2012) made an attempt. The applicability of algorithm in determining the various levels of bug severity for bug repositories has been validated using various performance measures by applying fivefold cross validation. They observed that the performance of machine learning techniques stabilizes as we increase the number of terms. F-measures of the data sets for severity level 2, 3 and 4 are more than 80 and 90 % for all most all the techniques. Based on the models developed, the severity level for the newly submitted bug reports can be predicted which will be helpful in automatic determination of severity level.

Abdelmoez et al. (2012) used Naive Bayes (NB) classifier to compute prediction model that will distinguish the very fast and the very slow bugs in order to prioritize which bugs to start with and which to exclude at the mean time respectively. They take into consideration the development effort coordination and addressed two questions that are which bug to fix first and how long it will take to fix. They gathered data from four software systems, using 17 attributes only from every bug report of the chosen systems, as they are the most commonly used attributes. Procedure of their work is they first obtain bug report information and than compute using NB algorithm. NB shows with better prediction performance.

Meanwhile, Dommati et al. (2012) focuses on the feature extraction, noise reduction in data and classification of network bugs using probabilistic Naive Bayes approach's different event models like Bernoulli and Multinomial on the extracted features. New unseen bugs are given as input to the algorithms, the performance comparison of different algorithms is done on the basis of accuracy and recall parameters. They concluded from the results that there is need to go into semantics of bug information. They had successfully extracted some of the bug specific features. By analyzing and depending on the static analysis of the bug reports, the feature extraction and selection has been performed.

This time in the next part of their work, Kanwal and Maqbool (2012) proposed and evaluated a classification based approach to build bug priority recommender. They used two classifiers, and presented a comparison to evaluate which classifier performs better in terms of accuracy. Another evaluation that they performed was to determine the combination of features that better determines the priority of a bug. They also proposed two new measures, Nearest False Negatives (NFN) and Nearest False Positives (NFP), which provide insight into the results produced by precision and recall. The main contributions were proposing and evaluating a classification based approach for automatic bug priority prediction, exploring different features, defining new measures for the evaluation of bug priority recommender. The highest accuracy is achieved with SVM when categorical and text features are combined for training.

Sharma et al. (2012) evaluated the performance of different machine learning techniques in predicting the priority of the newly coming reports on the basis of different performance measures. They use summary feature of bug report to predict the priority of newly coming bug report. Also, they used series of operators in Rapid Miner for pre-processing of the bug reports like stop words removal and tokenization. Evaluation of model has been done by applying cross project validation for 76 cases of five data sets of Open office and Eclipse projects. The accuracy of different machine learning techniques in predicting the priority of a bug report with in and across project is found better except Naive Bayes.

Thung et al. (2012) analyzed bugs from three Java software systems. They extracted bug reporting data from version control repositories and bug tracking systems, identify bug locations based on bug fixes, and back-trace bug introducing time based on change histories of the buggy code. Also, they removed nonessential changes, and most importantly, recover root causes of bugs from their treatments/fixes. Then they calculated the bug reporting latencies, and found that bugs have diverse reporting latencies. Based on the calculated reporting latencies and features they extracted from bugs, they build classification models that can predict whether a bug would be reported early (within 30 days) or later, which may be helpful for prioritizing bug fixing activities. Their evaluation on the three software systems shows that their proposed bug reporting latency prediction models could achieve an AUC (Area Under the Receiving Operating Characteristics Curve) of 70.869 %.

To automatically predict the severity of bug reports a new approach leveraging information retrieval based nearest neighbors in particular BM25-based document similarity function was proposed by Tian et al. (2012). Their approach automatically analyzes bug reports reported in the past along with their assigned severity labels, and recommends severity labels to newly reported bug. They focused on predicting fine-grained severity labels, namely the different severity labels of Bugzilla including: blocker, critical, major, minor, and trivial. Compared to the existing state of the art study on fine-grained severity prediction, the proposed approach brings significant improvement.

Xuan et al. (2012) have addressed the problem of the developer prioritization, which aims to rank the contributions of developers. They mainly explored two aspects, namely modeling the developer prioritization in a bug repository and assisting predictive tasks with their model. First, they modeled how to assign the priorities of developers based on a social network technique. Second, they consider leveraging the developer prioritization to improve three predicted tasks in bug repositories, i.e., bug triage, severity identification, and reopened bug prediction. Three problems are investigated, including the developer rankings in products, the evolution over time, and the tolerance of noisy comments. They empirically investigate the performance of proposed model and its applications in two bug repositories. The results indicate that the developer prioritization can provide the knowledge of developer priorities to assist software tasks, especially the task of bug triage.

An approach to predict the priority of a reported bug using different machine learning algorithms presented by Alenezi and Banitaan (2013). Also, they investigated the effect of using two feature sets (textual contents and meta data information of bug reports) on the classification accuracy. They conduct experimental evaluation using two open-source projects. The contributions of their work includes investigating the effectiveness of applying several machine learning techniques, evaluating the impacts of using different feature sets to build the predictive model and conduct experimental evaluation using two bug reports data sets. They used the meta-data features like component, operating system and severity because they contain useful information that may help in discriminating between priority levels.

Zanetti et al. (2013) proposed an efficient and practical method to identify valid bug reports which (a) refer to an actual software bug, (b) are not duplicates and (c) contain enough information to be processed right away. Their classification is based on nine measures to quantify the social embeddings of bug reporters in the collaboration network. They demonstrated its applicability in a case study, using a comprehensive data set of more than 700,000 bug reports obtained from the Bugzilla installation of four major OSS communities, for a period of more than 10 years. Based on this finding, they developed an automated bug report classification mechanism. They used nine topological measures at the level of bug reporters.

For reducing human efforts, for analyzing bug reports and supporting bug tracking system, Behl et al. (2014) presented an efficient bug classification tool in the process of identifying bug reports as security or non-security. They focuses on security bug and presents a bug mining system for the identification of security and non-security bugs using the term frequency-inverse document frequency (TF-IDF) weights and Naive Bayes. They performed experiments on bug report repositories of bug tracking systems. In a Probabilistic Naive Bayes they used different models like Bernoulli or Multinomial event model for classification purposes. Thereby, making TF-IDF based bug Mining tool feasible to use as a complimentary tool in the bug tracking system.

Blocking bugs are software bugs that prevent other bugs from being fixed. These blocking bugs may increase maintenance costs, reduce overall quality and delay the release of the software systems. To deal with this, recently, Garcia and Shihab (2014) build prediction models based on decision trees to predict whether a bug will be a blocking bug or not. As a data set, authors used 14 factors extracted from the bug repositories of six large open source projects. They analyzed these decision trees in order to determine which factor best indicates these blocking bugs. The goal is to help developers identify these blocking bugs early on. Their results show that proposed prediction models achieve better F-measures. They also find that the most important factors in determining blocking bugs are the comment text, comment size, the number of developers in the CC list of the bug report and the reporters experience. They found that blocking bugs take approximately two to three times longer to be fixed compared to non-blocked bugs. Their analysis shows that proposed models reduce the median time to identify a blocking bug.

A novel way of assigning software bug priority using supervised classification on clustered bugs data was presented by Goyal et al. (2015). Their work based on the study which claims that when classification is done on the data which is previously clustered, it significantly improves its performance. It is being proposed, to cluster the software bugs based on their similarity before directly applying classification algorithms on them. In this work, this approach has been used for the first time for predicting the priority of the software bugs to find if classifier performance improves when it is preceded with clustering. Using this system, clustering was performed on problem title attribute of the bugs to group similar bugs together using clustering algorithms. Classification was then applied to the clusters obtained, to assign priority to the bugs based on their attributes severity or component using classification algorithms. It was then studied which combination of clustering and classification algorithms used provided the best results. Overall, the best results were obtained with X Means, when used with the Bayesian Networks classifier.

An overview of above discussed researchers work in terms of techniques, evaluation criteria and data sets utilized is presented in Table 6. While comprehensive summary which includes features, results, pros and cons of these classification and information retrieval techniques are shown in Table 7.

Table 6 Overview of classification and information retrieval techniques

Researcher	Proposed technique	Compared techniques	Evaluation metrics	Datasets
Podgurski et al. (2003)	Cluster Analysis + Multivariate Visualization	Multivariate Visualization	HMDS display	GCC, Jikes, and javac
Yu et al. (2010)	Artificial Neural Networks	Bayes Approach	Recall, precision and F-measure	RIS2.0
Jaweria Kanwal (2010)	Support Vector Machine	–	Precision, Recall	Eclipse
Lamkanfi et al. (2010)	Naive Bayes	–	Precision, Recall	Mozilla, Eclipse, Gnome
Lamkanfi et al. (2011)	Naive Bayes Multinomial	Naive Bayes, K nearest Neighbor, Support Vector Machine	Accuracy, Set size	Eclipse, Gnome
Chaturvedi and Singh (2012)	Naive Bayes, K nearest Neighbor, Support Vector Machine, Naive Bayes Multinomial, J48, RIPPER	–	Precision, Recall, F-Measure, Accuracy	NASAs PITS projects
Abdelmoez et al. (2012)	Naive Bayes	–	Precision, recall	Mozilla, Eclipse, Gnome
Dommati et al. (2012)	Naive Bayes Multinomial, Naive Bayes Bernoulli	Naive Bayes	Precision, Recall, Accuracy and F-Measure	A networking based organization site
Kanwal and Maqbool (2012)	Support Vector Machine	Naive Bayes	Precision, Recall, NFN, NFP	Eclipse
Sharma et al. (2012)	Support Vector Machine, K nearest Neighbor, Neural net,	Naive Bayes	Accuracy, Precision and F measure	Open office, Eclipse
Thung et al. (2012)	Support Vector Machine	ADTree, Naive Bayes, Voted Perceptron	Operating Characteristic Curve	AspectJ, Rhino, and Lucene

Table 6 continued

Researcher	Proposed technique	Compared techniques	Evaluation metrics	Datasets
Tian et al. (2012)	Nearest Neighbors	SEVERIS	Precision, Recall and F measure	OpenOffice, Mozilla and Eclipse
Xuan et al. (2012)	Naive Bayes, Support Vector Machine	–	Precision, Recall, F-measure	Eclipse, Mozilla
Alenezi and Banitaan (2013)	Decision Tree, Random Forests	NB	Precision, Recall, F-measure	Eclipse, Firefox
Zanetti et al. (2013)	Support Vector Machine	–	Precision, Recall and F-Score	MozillaF, MozillaT, Eclipse, Nbeans
Behl et al. (2014)	TF-IDF	NB	Success, Precision	Bugzilla
Garcia and Shihab (2014)	Decision trees (C4.5)	Naive Bayes, kNN, Random Forests and Zero-R	Precision, Recall, F-measure and Accuracy	Chromium, Eclipse, FreeDesktop, Mozilla, NetBeans and OpenOffice
Goyal et al. (2015)	XMean, Expectation Maximization, Kmean + Bayes Net, Random Forest, Sequential Minimal Optimization	Bayes Net, Random Forest, Sequential Minimal Optimization	Accuracy, Precision and Recall	Software organization

Table 7 Comprehensive summary of classification and information retrieval techniques

Researcher	Results	Pros	Cons	Features used
Podgurski et al. (2003)	The results of applying proposed classification strategy to GCC, Jikes, and javac suggest that the strategy is effective and scales to large programs	Strategy performed well with a relatively simple type of classifier and with coarse-grained execution profiles. Approach may be useful for classifying computer intrusions reported by anomaly detection systems	Results may not generalize to other types of software. Hand crafted test inputs were used rather than operational inputs. Additional evaluation of the classification strategy is necessary	Textual and Categori-cal
Yu et al. (2010)	Based on the experiment results, the efficiency of Artificial Neural Networks model used in the defect priority prediction platform is better than the Bayes model	Proposing several strategies to increase the accuracy of the model based on the testing domain knowledge. The efficiency of troubleshooting is improved	Generalization ability may be reduced by over training	Categorical
Jaweria Kanwal (2010)	Among different combinations of bug report, precision and recall of Categorical, Summary and Long description(CSTF) category is better than all other categories in Support Vector Machine	Priority assignment assist triagers in resolving important bugs first. Both categorical and text and their different combinations were used	Precision of P1 priority level is less than 40 % for all feature categories. This means that many unimportant reports are assigned P1 priority	Textual and categorical
Lamkanfi et al. (2010)	Given a training set of sufficient size, it is possible to predict the severity with a reasonable accuracy where both precision and recall vary between 0.65 and 0.75	It is possible to predict the severity based on other information contained in a bug report, in particular the textual information describing the bug	Not guaranteed to hold with other software projects.The tools used to process the data might contain errors	Textual
Lamkanfi et al. (2011)	For the cases under investigation Naive Bayes Multinomial performs superior compared to the other proposed algorithms with respect to accuracy and training set size	It enables us to implement a more automated and more efficient bug triaging process. Bug report textual information can also be used to predict the severity.	The results obtained from presented approach are not guaranteed to hold with other software projects.The tools used to process the data might contain errors.Use bug reports submitted by the community thus likely to be unreliable	Textual

Table 7 continued

Researcher	Results	Pros	Cons	Features used
Chaturvedi and Singh (2012)	It is observed that the performance of machine learning techniques stabilizes as we increase the number of terms.Best F-measure for almost every proposed technique	Severity level for the newly submitted bug reports can be predicted which will be helpful in automatic determination of severity level. Quite useful for the triager in automatic assignment of reported bug to the developer for its fix	Accuracy of Naive Bayes classifier does not get stabilize in most of the projects	Textual
Abdelmoez et al. (2012)	Naive Bayes algorithm as recommended to explore other models results better prediction performance. Prioritize which bugs to start with and which to exclude at the mean time	They used quartiles Q1, Q3 for binning instead of using the median only in order to categorize the bugs into very fast bugs so developers can start with and very slow bugs in order to exclude and defer them	Small data set, post submission data not taken	Categorical
Dommati et al. (2012)	The experimental results show that applying Bayesian model for classification of bugs using word information as feature is reliable for two classes. Bernoulli Model is giving good results when compared to the Multinomial model on the basis of evaluation parameters used	Bernoulli and Multinomial Models using bug specific give better accuracy compared to word information. Also they had successfully extracted some of the bug specific features	It does not give any proper reason for the classification and cannot be used for multi class classification	Textual and categorical
Kanwal and Maqbool (2012)	Findings reveal that support vector machine are better than the Naive Bayes algorithm for text, whereas for categorical, Naive Bayes performance is better than support vector machine. The highest accuracy is achieved with support vector machine when categorical and text are combined for training	Accuracy of the classifiers indicates that proposed priority recommender can help triagers. NFN and NFP are useful measures for evaluating the recommendations made by the classifier	As compared to support vector machine, Naive Bayes is not good in handling the high dimensionality of text	Categorical and textual
Sharma et al. (2012)	Support vector machine, neural net give overall higher accuracy in comparison of k nearest neighbors and Naive Bayes for all data set in predicting the priority of the newly coming reports on the basis of different evaluation measures	Historical data of other projects developed in open source environment is better priority predictor. Cross project validation for different cases are working with better accuracy level	Accuracy of Naive Bayes is the lowest of all. K nearest neighbor performance decreased by increasing the number of terms	Textual

Table 7 continued

Researcher	Results	Pros	Cons	Features used
Thung et al. (2012)	Support vector machine found to be the most accurate classification algorithm in the solution. Perceptron is the fastest algorithm followed by Naive Bayes, ADTree, and lastly support vector machine. Evaluation on the three software systems shows that proposed bug reporting latency prediction models could achieve high AUC	Build classification models that can predict whether a bug would be reported early (within 30 days) or later, which may be helpful for prioritizing bug fixing activities	Even though support vector machine has the best performance, it takes a long time to run	Categorical and text
Tian et al. (2012)	Compared to the existing state of the art study on fine-grained severity prediction, namely the work by Menzies and Marcus, proposed approach brings significant improvement especially on hard to predict severity labels	Solution to predict the severity labels of bug reports as severity labels are important for developers to prioritize bugs	Errors in the experiment, generalizabilty of their findings. Accuracy of the proposed approach can further improve	Categorical and textual
Xuan et al. (2012)	Analyze three problems of the developer prioritization, namely the characteristics in products, the evolution, and the tolerance of noises. The prediction based on the mixed vectors can obtain better precision, recall, and F-measure on bugs in Mozilla	The results indicate that the developer prioritization can provide the knowledge of developer priorities to assist software tasks, especially the task of bug triage	Difference of results in Eclipse is not significant	Categorical and Text
Alenezi and Banitaan (2013)	The experimental results showed that both Random Forest and Decision Tree gave better classification results than Naive Bayes. Also investigated the effect of using two feature sets on the classification accuracy	The experimental evaluation shows that the proposed approach is feasible in predicting the priority of bug reports	Feature set 2(meta data features) outperforms Feature set 1 (textual features) dramatically for both data sets	Text and meta data
Zanetti et al. (2013)	Proposed classification mechanism achieves a remarkably high accuracy across different communities. An automated classification scheme that can easily be integrated into bug tracking platforms and analyze its performance in the considered OSS communities	A contribution towards classification schemes that are highly accurate, yet simple enough to be of practical relevance. Highlights the potential of using quantitative measures of social organization in collaborative software engineering	Extension of this methodology to newly born communities remains a challenge	Categorical data

Table 7 continued

Researcher	Results	Pros	Cons	Features used
Behl et al. (2014)	Tool based on TF-IDF is giving better results than the tool based on Naive Bayes in terms of precision and success rate. Model changed the mislabeled bug reports with a high success rate	The tool helps in deciding the priorities of the incoming bugs. Shows the benefits of applying text mining and log frequency weights over Naive Bayes approach	Proposed tool need to deal with more categories of bug reports like Crashbugs, Cleanup bugs, Performance bugs and Regression bugs	Textual
Garcia and Shihab (2014)	Results show that proposed prediction models achieved F-measures of 15–42 %, which is a two- to four-fold improvement over the baseline random predictors. Top Node analysis shows that the most important factors to determine blocking bugs are the comments, comment-size, number of developers in the CC list and the reporters experience	Analysis shows that proposed models reduce the median time to identify a blocking bugHelp developers identify these blocking bugs early on	Achieved low recall values for some of projects	Categorical and textual
Goyal et al. (2015)	Overall increase in the performance was always there when classification was preceded by any of the above mentioned clustering algorithms. The best results were obtained with X Means clustering, when used with the Bayes Net classifier	Classification on clustered data significantly improves its performance. Save developers time and effort, ultimately leading to saving of the resources of the organization itself	Single feature used. Need to apply proposed approaches for multiple features	Textual

5.2 Categorization approaches

An automated method named SEVERIS (SEVERity ISsue assessment) algorithm presented by Menzies and Marcus (2008) to assists the test engineer in assigning severity levels to defect reports. SEVERIS always found good issue predictors with high f-measures. Their study shows that unstructured text might be a better candidate for generating severity assessments than the structured data. Steps taken for conversion of unstructured data to structural data are tokenization, stop word removal, stemming, employing TF-IDF and InfoGain. The case study results indicate that SEVERIS is a good predictor for issue severity levels, while it is efficient easy to use.

Several solutions to the challenging task of clustering software defect reports was presented by Rus et al. (2009). Their work has been motivated by belief that the rich information in software defect reports, which are generated during the testing phase in the form of textual reports, can be of great value. They proposed advanced methods for clustering defect reports that take advantage of the description and summary fields of the reports. The experiments on defect reports from Mozillas Bugzilla and with three clustering algorithms showed that normalized cut using a TF-IDF vectorial representation based on a combination of descriptions and summaries of reports leads to better clustering than using the summary or the description of defects alone.

As an example of classification by clustering Nagwani and Verma (2011) used Suffix Tree Clustering(STC) algorithm for software bug classification. First clusters are created from the bug repositories and then labels are assigned to the each cluster, which indicates the classes of the clusters. Three software bug repositories are taken for experiment with different number of software bug records. Here STC implementation is available as the part of Carrot2 framework (component based framework for text clustering). They evaluated designed technique using the common clustering parameters. STC appears to be an effective way of classifying the software bug in just a small time, also cluster purity calculated is adoptable.

A software bug classification algorithm, CLUBAS (Classification of Software Bugs Using Bug Attribute Similarity) which is another example of classification by clustering was presented by Nagwani and Verma (2012). The proposed algorithm works in three major steps that are creation of text clusters, generation of cluster labels and mapping of the cluster labels against the bug taxonomic terms to identify the appropriate categories of the bug clusters. The designed algorithm is evaluated using the performance parameters that are then compared with the standard classification techniques using clustering algorithms. A GUI (Graphical User Interface) based tool is also developed in Java for the implementation of CLUBAS algorithm.

Similarly, Somasun and Murphy (2012) investigated whether combining Latent Dirichlet Allocation(LDA) with a machine learning approach could improve the consistency with which component recommendation could be performed for bug reports. They considered broadening the consistency of the recommendations produced by an automatic approach by investigating three approaches to automating bug report categorization. They focused on the use of the long free-form discussion portion of Bugzilla bugs for both the training and testing of the classifiers. The experiments on three open source projects showed that an approach which combines LDA with Kullbach-Leibler Divergence LDA-KL) can produce recommendations with more consistency in recall values across all components of a system than previous approaches.

An automated technique for bug labeling using Term Frequency-Inverse Document Frequency (TF-IDF) and Latent Semantic Indexing (LSI) presented by Chawla and Singh (2014). They suggest a method which takes semantic information present in the bug report into consideration. Experimental study shows that there is improvement in results with the addition of semantically similar words obtained from LSI in conjunction with the terms extracted using TF-IDF. Main steps in the proposed procedures are selection, preprocessing, training and testing data. Using LSI along with TF-IDF, they achieved better accuracy for the polish bug reports and for security bug reports as compared to using TF-IDF alone. This work has shown improvement in automatic bug labeling by addition of semantically similar words.

An overview of above discussed researchers work in terms of techniques, evaluation criteria and data sets utilized is presented in Table 8. While comprehensive summary which includes features, results, pros and cons of the classification by clustering approaches are shown in Table 9.

5.3 Other approaches

A technique to rank error reports emitted by static program checking analysis tools was explored by Kremenek and Engler (2003). They developed the idea of z-ranking that uses frequency counts of successful and failed checks to rank error messages from most to least probable. Z-ranking employs a simple statistical model to rank those error messages most likely to be true errors over those that are least likely. They demonstrated that z-ranking applies to a range of program checking problems and that it performs up to an order of magnitude better than randomized ranking. Further, it has transformed previously unusable analysis tools into effective program error finders. The authors explored the hypotheses comprise of probable and improbable error reports.

Meanwhile, Kim and Ernst (2007a) prioritized warning categories by analyzing the software change history. The underlying intuition is that if warnings from a category are resolved quickly by developers, the warnings in the category are important. The work aggregates properties of warning instances to prioritize warning categories. They suggest that their technique will be most effective when the categories are relatively fine-grained and homogeneous. They ran bug finding tools on each development transaction of open source projects. Results indicate that different warning categories have very different lifetimes. Based on that observation, they proposed a preliminary algorithm for warning category prioritizing.

Proceeding further in their work this time Kim and Ernst (2007b) observed the warnings output by bug-finding tools for three subject programs. They proposed an automated *h*istory-based *w*arning *p*rioritization (HWP) algorithm that mines previous fix and warning removal experience that is stored in the software change history. The underlying intuition is that if warnings from a category are eliminated by fix-changes, the warnings are important. Their prioritization algorithm improved warning precision. They extended the previous work as they incorporate information regarding bug fixes instead of warning lifetime, proposed a prioritization algorithm rather than merely observing the varying lifetimes of warnings and evaluation by using the software change history.

In the same year, Giger et al. (2010), computed prediction models in a series of experiments with initial bug report data as well as post-submission information from three active open source projects. They investigated empirically the relationships between bug report attributes and the time to fix. Their objective is to compute prediction models that can be used to recommend whether a new bug should and will be fixed fast or will take more time for resolution. They examined in detail if attributes of a bug report can be used to build such a recommender system. They used decision tree analysis to compute and tenfold cross validation to test prediction models and explore prediction models in a series of empirical studies with bug report data of six systems of the three open source projects. Their study shows that incoming bug reports can be classified into fast and slowly fixed, post-submission

Table 8 Overview of categorization techniques

Researcher	Proposed technique	Compared techniques	Evaluation metrics	Data sets
Menzies and Marcus (2008)	SEVERIS	–	Precision, Recall, F measure	NASA Project, PITS
Rus et al. (2009)	Normalized and Size regularized Cut, k-means	–	Cluster purity, Accuracy, and Normalized mutual information	Mozilla
Nagwani and Verma (2011)	Suffix Tree Clustering	–	Purity, Number of Clusters, Entropy and Time	Mozilla, Jboss-Seam, MySQL
Nagwani and Verma (2012)	CLUBAS	Naive Bayes, Naive Bayes, J48, Support Vector Machine	Precision, F-measure, Accuracy	Android, JBoss, Mozilla, MySql
Somasun and Murphy (2012)	Support Vector Machine + Term Frequency Inverse Document Frequency, Latent Dirichlet Allocation, Kullback Leibler divergence	Support Vector Machine	Recall	Bugzilla(Eclipse Platform, Eclipse Mylyn and Mozilla)
Chawla and Singh (2014)	TF-IDF, Latent Semantic Indexing	TF-IDF, MNB	Accuracy	Google Chrome

Table 9 Comprehensive summary categorization approaches

Researcher	Results	Pros	Cons	Features used
Menzies and Marcus (2008)	SEVERIS found good with high precision, f-measures and recall. For data sets with more than 30 examples of high severity issues, SEVERIS always found good issue predictors	Unstructured data considered, little domain knowledge, reduce the number of dimensions/attributes	Lack of consistency in PITS, conclusions are biased, rules are self-certifying	Textual
Rus et al. (2009)	Promising results for clustering software defect reports using natural language processing and graph partitioning techniques. Also, normalized cut achieved the best performance in terms of average cluster purity, accuracy, and normalized mutual information. Using the union of the summary field and description field for clustering is better than using either the description field alone or summary field alone	Clustering defect reports can be very useful for prioritizing the testing effort and to better understand the nature of software defects	The accuracy and normalized mutual information of k-means are significantly lower than those of N cut on all three vectorial representations. SRcut performed poorly on this data set	Textual
Nagwani and Verma (2011)	They evaluated designed technique using the common clustering parameters. STC appears to be an effective way of classifying the software bug in just a small time, also cluster purity calculated is adoptable	It is a classification by clustering technique. Phrases are used both to discover and to describe the resulting groups	STC not use bug features other than text. Small data set considered	Textual
Nagwani and Verma (2012)	CLUBAS show stability and performs better than other in terms of F measure. Naive Bayes and Naive Bayes Multinomial performs better in terms of accuracy than other algorithms in few data sets	CLUBAS is an example of classification using clustering technique.A GUI based tool for software bug classification CLUBAS is presented	CLUBAS does not perform well accuracy wise for few data sets	Textual and Categori-cal

Table 9 continued

Researcher	Results	Pros	Cons	Features used
Somasun and Murphy (2012)	Support Vector Machines + TFIDIF and Support Vector Machines + Latent Dirichlet Allocation have high recall. Latent Dirichlet Allocation + Kullback Leibler divergence more consistent than Support Vector Machines + TFIDF and Support Vector Machines + LDA	Better consistency across all components for which bugs must be categorized	Only text feature used and that is free-form discussion field in a Bugzilla bug report which cannot be changed after the information is first entered. As some comments tend to reduce the accuracy of classification instead of improving it	Textual
Chawla and Singh (2014)	Experimental study shows that there is improvement in results with the addition of semantically similar words obtained from Latent semantic Indexing in conjunction with the terms extracted using TF-IDF	Labeling or categorizing bugs will help in assigning severity and priority, predicting appropriate developer, doing postmortem analysis for prevention of bugs	The results may vary if applied on other repositories	Textual

data of bug reports improves prediction models. Assignee, reporter, and monthOpened are the attributes that have the strongest influence on the fix-time of bugs.

Tian et al. (2013) proposed an automated approach DRONE (PreDicting PRiority via Multi-Faceted FactOr ANalysEs) which enhances linear regression with their threshold approach to handle imbalanced bug report data. Based on machine learning that would recommend a priority level using information available in bug reports. Their approach considers multiple factors, temporal, textual, author, related-report, severity, and product, that potentially affect the priority level of a bug report. They investigated the an open bug repository and for comparison they experiment with several other classification algorithms. The result on a data set consisting of more than 100,000 bug reports from Eclipse shows that proposed approach outperforms the baseline approaches in terms of average F-measure.

Similarly, Saha et al. (2014) analyzed long lived bugs with five different perspectives: their proportion, severity, assignment, reasons, as well as the nature of fixes. Authors pointed out that analyzing entire bug data sets using various machine learning or data mining techniques is not sufficient in understanding long lived bugs due to the imbalanced data set, i.e., containing relatively low proportion of long lived bugs compared to others. They showed that although the software development and maintenance processes have advanced a lot, but there are still a significant number of bugs in each project that survive for more than a year. Study on four open-source Eclipse projects showed that there were a considerable number of long lived bugs in each system and over 90 % of them adversely affect the users experience. Authors concluded that the reasons of these long lived bugs are diverse including long assignment time and not understanding their importance in advance.

The empirical study on bug report field reassignments in open-source software projects is performed by Xia et al. (2014). To better understand why bug report fields are reassigned, authors manually collected bug reports that had their fields reassigned. They emailed bug reporters and developers asking why these fields got reassigned. Then, they performed a large-scale empirical study on types of bug report field reassignments in 4 open-source software projects. In particular, they investigated (1) number of bug reports whose fields get reassigned, (2) the difference in bug fixing time between bug reports whose fields are reassigned and not reassigned, (3) the duration a field gets reassigned, (4) the number of fields that get reassigned, (5) the number of times a field gets reassigned, and (6) whether the experience of bug reporters affect the reassignment of bug report field. Authors found that a large number (approximately 80 %) of bug reports have their fields reassigned, and the bug reports whose fields get reassigned require more time to be fixed than those without field reassignments.

A novel method for semi-automatic bug triage and severity prediction using topic model and multi-feature is proposed by Yang et al. (2014). First, they extracted topic(s) from historical bug reports in the bug repository and find bug reports related to each topic. Then they utilize multi-feature to identify corresponding reports that have the same multi-feature (e.g., component, product, priority and severity) with the new bug report. Thus, given a new bug report, they are able to recommend the most appropriate developer to fix each bug and predict its severity. To evaluate proposed approach, they not only measured the effectiveness of their study by using 30,000 golden bug reports extracted from three open source projects but also compared some related studies. These 30,000 bug reports were manually extracted from all 16, 462, 18 bug reports and named as golden bugs because they contain much more information.

An overview of above discussed researchers work in terms of techniques, evaluation criteria and data sets utilized are presented in Table 10. While comprehensive summary which includes features, results, pros and cons of the other approaches are shown in Table 11.

Table 10 Overview of other approaches

Researcher	Proposed technique	Compared techniques	Evaluation metrics	Datasets
Kremenek and Engler (2003)	Z-Ranking	Randomized Ranking	False positive rates, time and error rate	Linux, Company X
Kim and Ernst (2007a)	Preliminary algorithm	FindBugs, JLint, and PMD	Lifetime	Columba and jEdit
Kim and Ernst (2007b)	Prioritization Algorithm	FindBugs, JLint, and PMD	Precision, Recall and False positive rate	Columba, Lucene, and Scarab
Giger et al. (2010)	Decision Tree	Random Classification	Precision, Recall, summary statistic	Eclipse, Mozilla, Gnome
Tian et al. (2013)	PreDicting PRiority via Multi-Faceted FactOrANalysEs-DRONE(GRAY)	Multi Class, RIPPER, Naive Bayes Multinomial	Precision, recall, and F-measure	Eclipse
Saha et al. (2014)	Empirical Study Analysis	–	Time	Eclipse product (JDT, CDT, PDE, Platform)
Xia et al. (2014)	Empirical Study Analysis	–	Mean, median, max and min time	Openoffice, Netbeans, Eclipse, Mozilla
Yang et al. (2014)	Topic Model, Multi-Feature	Navie Bayes, k Nearest Neighbors	Accuracy, Severity, Precision, Recall, F-measure and MRR	Eclipse, Mozilla, Netbeans

To conclude, researchers have focused on automated approaches for prioritization of bug reports. Mostly researchers have addressed bug prioritization issue using techniques of data mining, like classification, information retrieval and text mining approaches. Few of them used the idea which claim that when classification employed on the data which is previously clustered; it gives significantly better results than using classification techniques alone (Goyal et al. 2015). Also our findings reveal that the results of some classifiers [like, Support vector Machine (SVM)] show better results for text features, whereas some [like, Naive Bayes (NB)] for categorical features, but the performance can be better when categorical and text features are combined (Kanwal and Maqbool 2012).

6 Significant results

In this section we list the significant facts that were found in this survey. The significant facts are prominent techniques, evaluation metrics, data sets, researchers and venues.

6.1 Significant techniques used

To resolve the issue of bug prioritization researchers used mostly the classification techniques and the most prominent were Naive Bayes (NB), Support Vector Machine (SVM) and k-Nearest Neighbors (kNN) as can be seen in Fig. 4. Researchers also used Decision Tree (DT),

Table 11 Comprehensive summary of other approaches

Researcher	Results	Pros	Cons	Features used
Kremenek and Engler (2003)	z-ranking applies to a range of program checking problems and that it performs up to an order of magnitude better than randomized ranking. Z-ranking found 3–7 times more bugs than the average number of bugs found by random ranking for the checkers they analyzed	Z-ranking would be useful in many static error checking tools. Z-ranking provides a simple way to control the impact of analysis approximations	The clustering of errors, however, causes the strong hypothesis not to hold. This appears to have a significant effect on z-rankings performance	Categorical
Kim and Ernst (2007a)	They computed the lifetime of each warning category, and found that some warning categories have a short lifetime, while others have a long lifetime and propose prioritization of each warning category based on its observed lifetime.Technique will be most effective when the categories are relatively fine-grained and homogeneous	Prioritizing warning categories based on their lifetimes may help to make bug detection tools more useful.This is a generic prioritization approach applicable to any bug finding tools	It does not give different priorities to two different instances from the same category	Categorical
Kim and Ernst (2007b)	Proposed prioritization algorithm improves warning precision to 17, 25, and 67 % respectively. Over 90 % of warnings remain in the program or are removed during non-fix changes	This new, program-specific prioritization effectively directs developers to errors. Prioritization algorithm increases the precision of warnings significantly	The subject programs might not be representative. Bug fix data is incomplete and some revisions are not compilable	Categorical
Giger et al. (2010)	Decision tree perform significantly better in precision, AUC and recall than Random Classification. They also show that the inclusion of post-submission bug report data of up to 1 month can further improve prediction models	Model is Useful to new developers or bug reporters as they give an insight in how bugs are prioritized in a software project	Applicability of these models to develop fully automated recommender systems is questionable	Categorical

Table 11 continued

Researcher	Results	Pros	Cons	Features used
Tian et al. (2013)	Experiments on more than a hundred thousands bug reports from Eclipse show that DRONE can outperform baseline approaches in terms of average F-measure by a relative improvement of 58.61 %	An automated approach based on machine learning that would recommend a priority level based on information available in bug reports	Inaccuracies in the threshold process on the final result of DRONE	Categorical and Textual
Saha et al. (2014)	Results indicate that the overall bug fixing time of many, if not all, long lived bugs can be reduced through careful prioritization. Findings also indicate that although there are a number of tools for supporting bug triaging and fixing they appear to realize very few benefits from them	Results will help both developers and researchers to better understand factors behind delays, improve the overall bug fixing process, and investigate analytical approaches for prioritizing bugs	Information in these systems may not be completely accurate.There might have been some unintentional misinterpretations during the manual verification. Findings may not be generalizable	Categorical
Xia et al. (2014)	They find that a large number (approximately 80 %) of bug reports have their fields reassigned, and the bug reports whose fields get reassigned require more time to be fixed than those without field reassignments	This study could help developers comprehensively understand bug report field reassignment	Experience of reporters could limit affect the bug report field reassignment. Bug report field reassignments could cause a delay in the bug fix	Textual and Categori-cal
Yang et al. (2014)	The results show that proposed approach is likely to effectively recommend the appropriate developer to fix the given bug and predict its severity. Proposed novel Approach outperforms other approaches like in terms of evaluation metrics used	In their evaluation result, they improved bug triage performance and severity prediction compared to other studies	They are not sure that our approach is also effective in commercial projects. Some noise (e.g., fake bug reports) may have affected the results	Textual and categori-cal

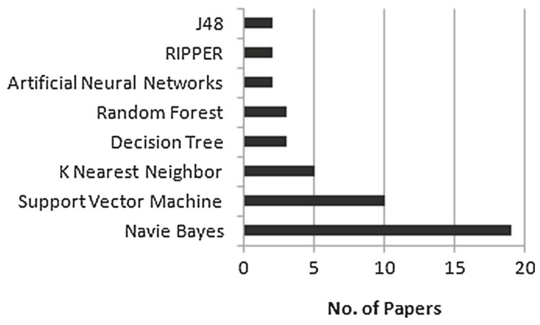


Fig. 4 Significant technique used

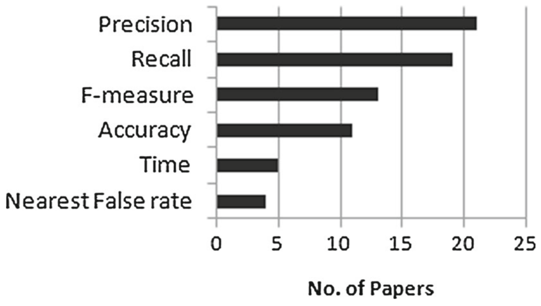


Fig. 5 Significant evaluation metrics used

Random Forests (RT), Artificial Neural Network (ANN), Repeated Incremental Pruning to Produce Error Reduction (RIPPER) and J48 (Java Implementation of C4.5) classifiers.

6.2 Significant evaluation metrics used

Figure 5 presents the top six evaluation metrics used by researchers to validate their results. Which shows that precision and recall have used highest number of times. Other evaluation criteria were F-measure, accuracy, time and false positive rate.

6.3 Significant data set used

The commonly used data sets that researchers used for purpose of bug prioritization can be seen in Fig. 6. We can see that most significant data sets were Eclipse and Mozilla repositories. Whereas as other researchers used Gnome, Net beans and Open office etc. It has also explored that almost all data sets are taken from open source repositories.

6.4 Significant venues

Figure 7 shows a pie chart that presents the number of publications published in different venues. According to our survey, only IEEE and ACM have published significant work on bug prioritization. Whereas venues like Science Direct, Wiley, IET that were considered in the start of systematic literature survey but to the best of our knowledge no work so far had been published by them. Remaining work on bug prioritization that published on other venues like Springer, book chapters and other conference proceedings were collected by using Google scholar and included in category of others.

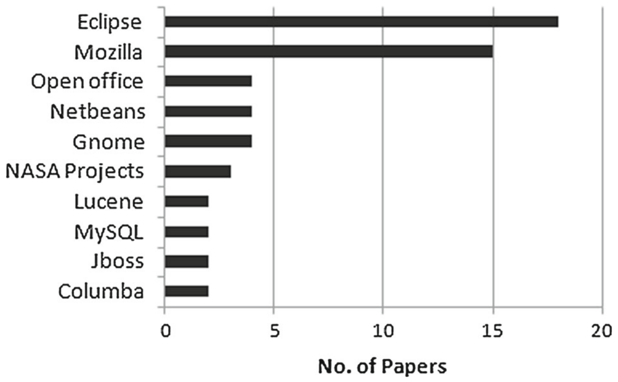


Fig. 6 Significant data set used

Fig. 7 Significant venues

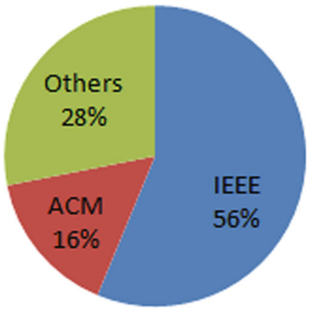
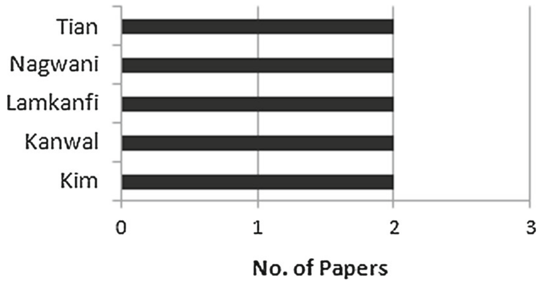


Fig. 8 Significant researchers



6.5 Significant researchers

Kim, Lamkamfi, Kanwal, Nagwani and Tian contributed equally to enrich this field. So, they are considered as prominent researchers who worked on bug prioritization as can be seen in Fig. 8.

7 Discussion-gaps

Many studies about bug prioritization receive attention of the researchers. Although, many success have been achieved, however, there are still missing gaps that need to be filled. We summarize some of the gaps about bug prioritization process, as follows:

1. Many researchers have focused on automated bug-report triage using machine learning methods. The problem with traditional supervised machine learning methods is that these methods require large amount of labeled data for training the classifier. Since labeled data is difficult to obtain, so, need extensive effort and expensive to process (Nigam et al. 2012).
2. Though, classifiers performance is improved when it is applied on clustered bug data sets but the issues with this approach are (1) Goyal et al. (2015) have considered only single feature for validation, (2) Also, accuracy concern (Nagwani and Verma 2012), and (3) Cluster purity for large data set (Nagwani and Verma 2011) need to be further investigated. For that, Karaboga and Ozturk (2011), Naseem et al. (2013) claims meta heuristics and cooperative clustering approaches can be better alternative for clustering and classification approaches.
3. Appropriate feature selection is baseline for classification task. Bug classifiers performance vary for textual, categorical or combination of both features (Kanwal and Maqbool 2012). Thus, more work should be conducted to develop more general appropriate feature selection to aid bug prioritization. To resolve the issue of vagueness and uncertainty in bug data one can use mathematical tool like Rough set theory which is considered as best for such issues (Pawlak et al. 1995).
4. In software development, one cannot afford to provide incorrect priority to the bugs. Although, using proposed machine learning techniques have provided good results, but still there is a scope for improvement in terms of accuracy, precision, recall and F-measure. This is because, researchers on bug-report triage may focus on improving the accuracy of bug-report triage by using some new clustering with classification approaches to improve the performance of bug classifiers (Goyal et al. 2015). None of the existing approach has achieved satisfactory accuracy (e.g., more than 95 %) (Zhang et al. 2015). Due to the accuracy concern, it is hard to apply existing automatic bug-report triage approaches in practice. Machine learning and tossing graphs can be tried for bug prioritization as they have proven to be promising for automated developer assigning for bug reports Bhattacharya et al. (2012) and Jeong et al. (2009).
5. Research is also needed to investigate the industrial case studies or close source projects and to apply the existing approaches on comparatively larger data sets (Alenezi and Banitaan 2013). And also there is need to refine the evaluation criteria to obtain better picture of strength and weakness of various techniques.

8 Conclusion and future work

When a bug tracking system receives a new filed bug report, the triager makes decisions about several characteristics of bug reports such as priority and severity levels. The bug priority level indicates the importance of that bug from business perspective. It gives an indication of the order in which bug reports should be fixed. Handling these reports manually is time consuming, and often results in delaying the resolution of important bugs. To address this issue, a recommender may be developed which automatically prioritizes the new bug reports with high accuracy. There is reasonable volume of research on bug-report prioritization.

This paper summaries the existing literature which covers a range of work in the field of bug triaging that specifically deal with bug prioritization. In this survey, we first briefly discussed some preliminaries. Second, we discussed the related surveys about bug reports to compare our survey. Third, through systematic literature survey method some statistics on bug prioritization research to show the amount of work on it was presented. Fourth, a rather thorough survey on existing bug-report prioritization work was conducted and each researcher work was explained in detail with pros and cons of all approaches. Fifth, different types of significant outcomes of our survey were presented. We finally discussed some gaps and questions that should be completed for further research.

This study focus on bug prioritization however, our future plan is to explore new strategies and algorithms to improve bug triaging system. Moreover, research gaps presented in the study is the evidence of research need to enhance the performance of well-known classification algorithms.

Author contributions Authors whose names appear on the submission have contributed sufficiently to the scientific work and therefore share collective responsibility and accountability for the results.

Compliance with ethical standards

Informed consent Consent to submit has been received explicitly from all co-authors, as well as from the responsible authorities—tacitly or explicitly—at the institute/organization where the work has been carried out, before the work is submitted.

References

Abdelmoez W et al (2012) Bug fix-time prediction model using Naïve Bayes classifier. In: 22nd International conference on computer theory and applications, IEEE, October, pp 167–172. doi:10.1109/ICCTA.2012.6523564

Alenezi M, Banitaan S (2013) Bug reports prioritization: which features and classifier to use? In: 12th International conference on machine learning and applications, IEEE, pp 112–116. doi:10.1109/ICMLA.2013.114

Anvik J et al (2005) Coping with an open bug repository technical report UBC-CS-TR-2005-20. In: Proceedings of the OOPSLA workshop on eclipse technology eXchange. ACM, pp 1–5

Anvik J et al (2006) Who should fix this bug? In: Proceedings of the 28th international conference on software engineering, pp 361–370

Anvik J, Murphy GC (2011) Reducing the effort of bug report triage. ACM Trans Softw Eng Methodol 20(3):1–35. doi:10.1145/2000791.2000794

Behl D et al (2014) A bug mining tool to identify and analyze security bugs using Naive Bayes and TF-IDF. In: International conference on reliability optimization and information technology, IEEE, pp 294–299. doi:10.1109/ICROIT.2014.6798341

Bennett K, Rajlich V (1999) Software maintenance and evolution: a roadmap. In: Proceeding ICSE'00 proceedings of the conference on the future of software engineering, pp 73–87

Bhattacharya P, Neamtiu I, Shelton CR (2012) Automated, highly-accurate, bug assignment using machine learning and tossing graphs. J Syst Softw 85(10):2275–2292. doi:10.1016/j.jss.2012.04.053

Brereton P et al (2007) Lessons from applying the systematic literature review process within the software engineering domain. J Syst Softw 80:571–583. doi:10.1016/j.jss.2006.07.009

Chapin N (2000) Software maintenance Types-A fresh view. In: International conference on software maintenance, pp 247–252. doi:10.1109/ICSM.2000.883056

Chaturvedi KK, Singh VB (2012) Determining bug severity using machine learning techniques. In: CSI sixth international conference on software engineering, IEEE, pp 1–6. doi:10.1109/CONSEG.2012.6349519

Chawla I, Singh SK (2014) Automatic bug labeling using semantic information from LSI. In: Seventh international conference on contemporary computing, IEEE, pp 376–381. doi:10.1109/IC3.2014.6897203

Cubranic M (2004) Automatic bug triage using text categorization. In: International conference of software engineering and knowledge engineering, pp 1–6

Dommati SJ et al (2012) Bug classification: feature extraction and comparison of event model using Naïve Bayes approach. In: International conference on recent trends in computer and information engineering, pp 8–12

Garcia HV, Shihab E (2014) Characterizing and predicting blocking bugs in open source projects categories and subject descriptors. In: Proceedings of the 11th working conference on mining software repositories, pp 72–81

Giger E et al (2010) Predicting the fix time of bugs. In: Proceedings of the 2nd international workshop on recommendation systems for software engineering—RSSE. ACM Press, New York, NY, USA, pp 52–56. doi:10.1145/1808920.1808933

Goyal N et al (2015) Advances in intelligent informatics, advances in intelligent systems and computing, vol 320. Springer International Publishing, Cham. doi:10.1007/978-3-319-11218-3

Guo PJ et al (2010) Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. In: Proceedings of the 32nd ACM/IEEE international conference on software engineering, pp 495–504

Herraiz I et al (2008) Towards a simplification of the bug report form in eclipse. In: Proceedings of the international working conference on mining software repositories, pp 145–148

Hooimeijer P, Weimer W (2007) Modeling bug report quality categories and subject descriptors. In: Proceedings of the twenty-second IEEE/ACM international conference on automated software engineering, pp 34–43

Jalbert N, Weimer W (2008) Automated duplicate detection for bug tracking systems. In: IEEE international conference on dependable systems and networks With FTCS and DCC (DSN) pp 52–61. doi:10.1109/DSN.2008.4630070

Jaweria Kanwal OM (2010) Managing open bug repositories through bug report prioritization using SVMs. In: Proceedings of the 4th international conference on open-source systems and technologies, ICOSST, pp 1–7

Jeong G, Kim S, Zimmermann T (2009) Improving bug triage with bug tossing graphs. In: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, pp 111–120

Jianhong Z et al (2010) A neural network based approach for modeling of severity of defects in function based software systems. In: International conference on electronics and information engineering, vol 2(Iceie), pp V2-568–V2-575. doi:10.1109/ICEIE.2010.5559743

Kanwal J, Maqbool O (2012) Bug prioritization to facilitate bug report triage. J Comput Sci Technol 27(2):397–412. doi:10.1007/s11390-012-1230-3

Karaboga D, Ozturk C (2011) A novel clustering approach: Artificial Bee Colony (ABC) algorithm. Appl Soft Comput 11(1):652–657. doi:10.1016/j.asoc.2009.12.025

Kaur M, Garg SK (2014) Survey on clustering techniques in data mining for software engineering. Int J Adv Innov Res 3(4):238–243

Kim S, Ernst MD (2007a) Prioritizing warning categories by analyzing software history. In: Fourth international workshop on mining software repositories, pp 27–27. doi:10.1109/MSR.2007.26

Kim S, Ernst MD (2007b) Which warnings should I fix first? Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering—ESEC-FSE, p 45. doi:10.1145/1287624.1287633

Kitchenham B, Charters S (2007) Guidelines for performing systematic literature reviews in software engineering. Engineering 2:1051. doi:10.1145/1134285.1134500

Kremenek T, Engler D (2003) Z-ranking: using statistical analysis to counter the impact of static analysis approximations. In: Static analysis. doi:10.1007/3-540-44898-5_16

Lamkanfi A et al (2010) Predicting the severity of a reported bug. In: 7th IEEE working conference on mining software repositories, IEEE, pp 1–10. doi:10.1109/MSR.2010.5463284

Lamkanfi A et al (2011) Comparing mining algorithms for predicting the severity of a reported bug. In: 15th European conference on software maintenance and reengineering, pp 249–258. doi:10.1109/CSMR.2011.31

Lazar A et al (2014) Generating duplicate bug datasets. In: Proceedings of the 11th working conference on mining software repositories, pp 392–395. doi:10.1145/2597073.2597128

Menzies T, Marcus A (2008) Automated severity assessment of software defect reports. In: IEEE international conference on software maintenance, pp 346–355. doi:10.1109/ICSM.2008.4658083

Mishra S, Kumar S (2015) Survey on types of bug reports and general classification techniques in data mining. Int J Comput Sci Inf Technol 6(2):1578–1583

Nagwani NK, Verma S (2011) Software bug classification using suffix tree clustering (STC) algorithm. Int J Comput Sci Technol 4333:36–41

Nagwani NK, Verma S (2012) CLUBAS: an algorithm and Java based tool for software bug classification using bug attributes similarities. J Softw Eng Appl 5(6):436–447. doi:10.4236/jsea.2012.56050

Naseem R et al (2013) Cooperative clustering for software modularization. J Syst Softw 86(8):2045–2062. doi:10.1016/j.jss.2013.03.080

Nigam A et al (2012) Classifying the bugs using multi-class semi supervised support vector machine. In: International conference on pattern recognition, informatics and medical engineering, pp 393–397. doi:10.1109/ICPRIME.2012.6208378

Pawlak Z et al (1995) Rough sets. Commun ACM 38(11):88–95. doi:10.1145/219717.219791

Podgurski A et al (2003) Automated support for classifying software failure reports. In: Proceedings of 25th international conference on software engineering, vol 6, pp 465–475. doi:10.1109/ICSE.2003.1201224

Punitha K, Chitra S (2013) Software defect prediction using software metrics—a survey. In: International conference on information communication and embedded systems, pp 2–5

Rus V et al (2009) Towards architecture-centric collaborative software development. In: Zhurnal Eksperimental'noi i Teoreticheskoi Fiziki

Saha RK et al (2014) An empirical study of long lived bugs. In: Software evolution week—IEEE conference on software maintenance, reengineering, and reverse engineering, IEEE, pp 144–153. doi:10.1109/CSMR-WCRE.2014.6747164

Sharma M et al (2012) Predicting the priority of a reported bug using machine learning techniques and cross project validation. In: 12th International conference on intelligent systems design and applications (ISDA), IEEE, pp 539–545. doi:10.1109/ISDA.2012.6416595

Somasun D, Murphy GC (2012) Automatic categorization of bug reports using latent Dirichlet allocation. In: Proceedings of the 5th India software engineering conference, pp 125–130. doi:10.1145/2134254.2134276

Thung F et al (2012) When would this bug get reported? In: 28th IEEE international conference on software maintenance, IEEE, pp 420–429. doi:10.1109/ICSM.2012.6405302

Tian Y et al (2012) Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In: 19th Working conference on reverse engineering, pp 215–224. doi:10.1109/WCRE.2012.31

Tian Y et al (2013) DRONE: predicting priority of reported bugs by multi-factor analysis. In: IEEE International conference on software maintenance, IEEE, pp 200–209. doi:10.1109/ICSM.2013.31

Vans AM (1999) Program understanding behavior during corrective maintenance of large-scale software. Int J Hum Comput Stud 51:31–70

Xia X et al (2014) An empirical study of bug report field reassignment. In: Software evolution week—IEEE conference on software maintenance, reengineering, and reverse engineering, IEEE, pp 174–183. doi:10.1109/CSMR-WCRE.2014.6747167

Xuan J et al (2012) Developer prioritization in bug repositories. In: 34th International conference on software engineering (ICSE), IEEE, pp 25–35. doi:10.1109/ICSE.2012.6227209

Yang G et al (2014) Towards Semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In: IEEE 38th annual computer software and applications conference, IEEE, pp 97–106. doi:10.1109/COMPSAC.2014.16

Yu L et al (2010) Predicting defect priority based on neural networks. Adv Data Min Appl Lect Notes Comput Sci 6441:356–367

Zanetti MS et al (2013) Categorizing bugs with social networks: a case study on four open source software communities. In: 35th International conference on software engineering, IEEE, pp 1032–1041. doi:10.1109/ICSE.2013.6606653

Zhang J et al (2015) A survey on bug-report analysis. Sci China Inf Sci 58(2):1–24. doi:10.1007/s11432-014-5241-2

Zhang T, Lee B (2013) A hybrid bug triage algorithm for developer recommendation. In: Proceedings of the 28th annual ACM symposium on applied computing, p 1088. doi:10.1145/2480362.2480568