participated in the bug and the fix time. They created their own linear model, predicted fix time as a continuous value, and reported mean magnitude of relative error (MMRE) between 70% and 80%. Other studies chose to classify bugs in terms of *slow*, *fast* or very *slow*, *low*, fast, very *fast*. The categories vary depending on the selection of researchers in the context of projects. Zhang et al. [25] used kNN-based classification method by using the bug attributes to classify bugs as *slow* or *fast*, and got the accuracy of 72.5%. Giger et al. [13] used initial bug attributes at the time when a bug is first reported, and post-submission data as these attributes changed over time, to make a binary classification. According to their results, post-submission data improves the accuracy of their model by between 5% and 10%. They achieved an overall accuracy of 60% and 70%. Habayeb et al. [14] proposed different model than the previous ones, and used temporal activities of the developers (changes in severity, priority, code review, comment exchange, etc. are some examples of the activities) to classify the bug fix time.

In addition, there are some studies that focus on the multi-label classification. Marks et al. [15] used Random Forest Classifier, and got the accuracy rate 65% in predicting three bug fix categories (<3 Months, <1 Year, <3 Years), and they found that priority and severity have small effects on fix time prediction on open source projects. Abdelmoez et al. [1] used the initial bug data (attributes, such as reporter, component, priority, etc.) to classify as four categories (*very fast*, *fast*, *slow*, *very slow*). They concluded that developers should focus on the *very fast* bugs, and defer the *very slow* bugs. Panjer used the initial attributes of the bug to predict the fixing time by using rules generated by different algorithms (0-R, 1-R, C4.5 Decision Tree, Naive Bayes, Logistic Regression), and achieved an accuracy of 34.9%. He determined the most important factors are the severity and the content of the summary [17]. Vijayakumar and Bhuvaneswari are also analyzed the bug effort category (e.g. 0-6 hours in category A, 6-12 hours in B, etc.) with the attributes and made rule based classification. They concluded that attributes defined poorly correlated inversely with the effort category, and implied that for better effort category prediction, the bug report should be enhanced [22].

Some of the works use only textual bug information for predicting bug fix time. Zimmermann et al. [23] proposed to use textual description of the bug to determine the fixing effort of it, and according to their results the predictions are very close to the bug's actual fix time, for issue type bugs, they are off by only one hour. On the other hand, Ramarao et al. [18] took one step forward of using textual attributes by adding the newly defined reporter reputation attribute. They showed that using bug attributes with textual data improves the accuracy of the predictions. They achieved 65% classification accuracy. Inspired from these papers [20, 25], we follow a different approach during which bug attributes and textual information are used to build prediction models. We predict bug fix time in terms of four classes, as in the paper [1] such as *fast*, *very fast*, *slow*, and *very slow*.

III. DATASET

Bugzilla is a bug tracking tool, and its Application Programming Interface(API) is used to get bug related information. Using this API [8], we downloaded 1283233 bug reports starting from September 1994 until August 2017. Mozilla, one of the most popular open source projects, is selected as our dataset, and one component of the product is targeted to decrease the possible effects between the components. As for the component in which the highest number of issues were reported, JavaScript Engine is selected. In total 33840 bugs were reported for JavaScript Engine.

A. A Bug's Lifecycle in Bugzilla

Bugzilla is a free bug tracking tool that presents solution to thousands of organizations provided by the Mozilla [6]. It provides keep track of issues, enhancements, tasks, and more. It has many features, such as notifications controlled by the users, reports, charts, custom workflows, time tracking system, localization, etc. When a new bug is created in Bugzilla, it takes the status *UNCONFIRMED*, waiting for the Mozilla Quality Member for confirmation. It can move to status *RESOLVED*, taking the resolution values such as *DUPLICATE* (the bug already exists), *WONTFIX* (the bug can't be fixed), *WORKSFORME* (the bug could not be reproducible), *INVALID* (the created record is not a real bug), *INCOMPLETE* (the provided bug's information is not enough to reproduce). If the bug is confirmed, it goes to state *NEW*, then it can be assigned to one of the developers and take the status *ASSIGNED* or the bug can be resolved and takes the status *RESOLVED*, and resolution as *FIXED*. The developer works on the bug, and makes changes and deploys it to the main repository; now the bug is ready for QA (quality assurance) engineer. If QA checks the issue and could not reproduce it, the bug status goes to *VERIFIED*, otherwise the provided solution is not enough and the QA makes the bug status as *REOPENED*. After the status *VERIFIED*, the bug status goes to *CLOSED*. This progress can be reviewed in [7].

B. Bug Selection

Before starting our analysis, we decided that some bugs might have erroneous fields and hence, they should be filtered out of our dataset. From the bug histories, bugs being resolved only once are taken into account. For example, if a bug's status has changed from *RESOLVED* to *OPEN*, and then to *RESOLVED*, we did not include this bug into our dataset. This helped us avoid reopened bugs, whose fix time is significantly higher than the other bugs [9]. The bugs that have status *RESOLVED*, *VERIFIED*, *CLOSED* and resolution as *FIXED* are taken into consideration. Duplicate bugs are also ignored because it is unclear which bug has the real information. The bugs whose summary or description fields are empty are also ignored. There have been 33840 bugs in this component, after these filters, we get total of 15513 bugs, created between the years 1998 until 2017.

C. Extracting Fix Time

Normally, the fix time is defined as the time between a bug's creation and its last change. According to this approach, using last change time of a bug might be sometimes misleading. For example, the developer can solve the task and can forget to carry forward the bug status, so this leads the mistaken calculation. As reported in Table I, the time to change a bug's status from *RESOLVED* to *VERIFIED* took an average of 1772.30 hours within the 2229 bugs. This means although the fix was done