# Automatically Predicting Bug Severity Early in the Development Process

Jude Arokiam
Ontario Tech University
Oshawa, Ontario, Canada
jude.arokiam@ontariotechu.ca

Jeremy S. Bradbury
Ontario Tech University
Oshawa, Ontario, Canada
jeremy.bradbury@ontariotechu.ca

## ABSTRACT

Bug severity is an important factor in prioritizing which bugs to fix first. The process of triaging bug reports and assigning a severity requires developer expertise and knowledge of the underlying software. Methods to automate the assignment of bug severity have been developed to reduce the developer cost, however, many of these methods require 70-90% of the project's bug reports as training data and delay their use until later in the development process. Not being able to automatically predict a bug report's severity early in a project can greatly reduce the benefits of automation. We have developed a new bug report severity prediction method that leverages how bug reports are written rather than what the bug reports contain. Our method allows for the prediction of bug severity at the beginning of the project by using an organization's historical data, in the form of bug reports from past projects, to train the prediction classifier. In validating our approach, we conducted over 1000 experiments on a dataset of five NASA robotic mission software projects. Our results demonstrate that our method was not only able to predict the severity of bugs earlier in development, but it was also able to outperform an existing keyword-based classifier for a majority of the NASA projects.

## CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**; **Maintaining software**; *Software testing and debugging*; • **Computing methodologies** → *Machine learning*.

## KEYWORDS

bug severity, natural language processing, machine learning

**ACM Reference Format:**
Jude Arokiam and Jeremy S. Bradbury. 2020. Automatically Predicting Bug Severity Early in the Development Process. In *New Ideas and Emerging Results (ICSE-NIER'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3377816.3381738

## 1 INTRODUCTION

Even when using software development best practices, the occurrence of bugs in software is inevitable. When an incorrect or unexpected behaviour manifests itself as a bug, it is important for an organization to be able to quickly and efficiently triage and fix the bug. A common industry practice is to use bug reports to document and manage known bugs. Upon detection, a developer will submit a bug report that contains a bug description and any other information that is useful to fixing the bug. Once a bug report has been submitted, it is triaged and a bug severity is assigned (e.g., high, medium, low) – the assignment of a bug severity is a resource intensive task. For example, the Mozilla bug database contains hundreds of thousands of bug reports and receives 135 new reports per day [6]. Manually assessing these reports can be time consuming and requires knowledge of the software to ensure the correct bug severities are assigned. Furthermore, it increases an organization's developer cost to build and maintain their software systems [1].

To reduce development costs associated with bug report triage, automated approaches have been developed to predict bug severity using natural language processing (NLP) the presence of keywords in bug reports [2–4, 8, 10, 11]. These approaches commonly share a similar severity prediction process: (1) text mining is used to collect and pre-process bug reports; (2) a classifier is trained that uses a machine learning algorithm with bug reports that have already had their severity assigned; (3) the classifier is used to predict the severity of a new bug report. While the prediction process is often consistent, past approaches may vary based on the content mined from the bug reports and the machine learning algorithm used for training and prediction. A major drawback of past approaches is that the text mining is usually based on project-specific keywords (the content of the bug reports) and experiments have shown that 70-90% of a project's total bug reports are required to properly train the classifier [2, 8]. The need for project-specific data to effectively train the classifier greatly reduces the benefits of automation.

We believe that a practical solution to predicting bug report severity must be effective early in the software development process – from the first occurrence of a bug. Having an effective bug severity prediction method available earlier will allow more bug report assessments to benefit from automation and reduce costs. Furthermore, it can be argued that the prediction of bug report severity is the most beneficial at the early stages of a project as this is when developers have less expertise and insight with respect to the software under development. We have developed a method that satisfies the above requirement by leveraging an organization's common practices. For example, the bug reports of an organization's past projects should be written similarly to the reports that will be created for a new project. Our hypothesis is that: *How bug* *reports in past projects are written can be used to accurately predict the severity of a bug report at the beginning of a new project.*

In this paper we describe our new method to automatically predict the severity of a bug based on how bug report descriptions are written. We train our method on data from past projects, thus allowing bug severity prediction at the beginning of a new project. Finally, we evaluate our method and compare it with an existing keyword-based prediction approach. Our results demonstrate that our method outperforms the existing keyword-based approach in the majority of projects under evaluation.

## 2 BACKGROUND

Bug reports are used to keep track of bugs in a given software project. Developers submit these reports into a bug tracking system (e.g., JIRA[1], Bugzilla[2]). The structure and content of bug reports will vary based on organizational preferences and the bug tracking system used. Examples of bug report content include a summary, a bug description, code attachments and the name of the developer reporting the bug. The summary and bug description are considered the main indicators of predicting bug severity [3].

### 2.1 Bug Severity

Severity is *"the impact the bug has on the successful execution of the software system"* [3]. Severity ratings derived from bug reports help determine in what order bugs should be fixed (priority) and who should be assigned to fix each bug. Bug severity categories can vary based the organization and the bug tracking system being used:

- severe, non-severe;
- high, medium, low;
- trivial, minor, normal, major, critical, blocker; and
- ranges of numeric values.

### 2.2 Keyword-based Prediction of Bug Severity

Menzies and Marcus [8] developed SEVERIS that was designed as a human-in-the loop tool for providing severity rating recommendations for bug reports to test engineers. SEVERIS was evaluated using a dataset of bug reports from five NASA robotic missions stored in NASA's Project and Issue Tracking System (PITS) [7]. The features extracted from the bug reports were textual summary keywords and the classifier used was rule learning. We have reproduced the SEVERIS tool and study and used the results as a baseline for evaluating our prediction method (Section 4).

## 3 APPROACH

We have developed a new bug report severity prediction method[3] that builds on the strengths of existing methods like SEVERIS while also addressing the weaknesses. Our method requires no project-specific training data and can be used to predict the bug severity rating from the occurrence of the first bug. Furthermore our method is general and can be adapted to use different types of textual inputs. The general process (see Figure 1) of our method is as follows:

[1]https://www.atlassian.com/software/jira/
[2]https://www.bugzilla.org/
[3]Our tool is open source and available at *https://github.com/sqrlab/bug-severity-prediction.*

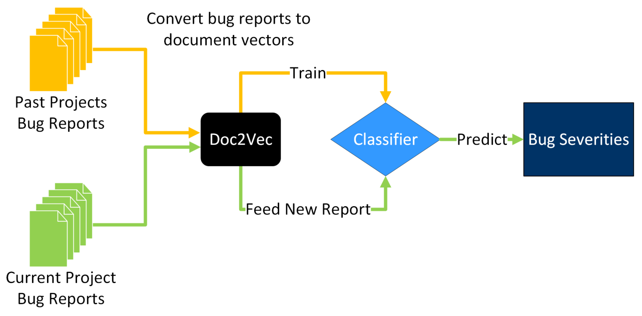
Convert bug reports to document vectors
Past Projects Bug Reports
Train
Doc2Vec
Classifier
Predict
Bug Severities
Feed New Report
Current Project Bug Reports

**Figure 1: Bug report severity prediction process**

(1) Extract bug summary and bug severity from bug reports in past projects
(2) Convert extracted summaries to document vectors
(3) Train classifier using past project data from the same organization
(4) Extract summary of a new bug report that has not yet been assigned a severity
(5) Convert summary to document vector
(6) Input the new document vector into the classifier to generate a bug severity prediction

While the use of cross-project data has been used in software defect prediction within source code [12], to the best of our knowledge, it has not been applied to bug severity prediction using natural language.

### 3.1 Input

The input data for the training phases of our method is the bug report text summaries from one or more past projects. The training input data will also have the assigned bug severity included. The input data for the testing phase of our method is the text summary for a single bug report in the current software project.

### 3.2 Pre-Processing and Feature Extraction

In order to use past project bug reports to train our classifier, we can no longer rely on project-specific keywords used in approaches like SEVERIS. Instead the features we use must be appropriate for cross-project data and contain information about how each bug report summary is written. To capture how a bug report is written, we decided to use Doc2Vec document vectors because document vectors retain sentence semantics and word ordering and also because creating a vector space of documents allows us to generalize [5]. Doc2Vec is a tool that converts text like phrases, sentences, paragraphs or even entire documents into a document vector [5] which is ideal for the textual summaries found in bug reports. Derived from Word2Vec, there are two available models of Doc2Vec: distributed memory (DM) and distributed bag of words (DBOW) [9]. Le and Mikolov [5] experimented on both models and concluded that that both models perform well for certain types datasets and should be chosen accordingly. Their recommendation was to combine both models for the best results and we have followed this advice.

**Table 1: Information contained in a NASA PITS bug report**

| Field | Format | Example |
|---|---|---|
| **TIM_Id** | String | ProjectA - TIM - 209 |
| **Subject (Summary)** | String | Uninitialized Variables |
| **Severity** | Numeric | 3 |
| **Description** | String | Filename sts_df.c |
| **Initiation Date** | Date | 3/25/07 |

**Table 2: Number of bug reports for the NASA PITS projects**

| Project ID | A | B | C | D | E |
|---|---|---|---|---|---|
| **Number of Bug Reports** | 965 | 987 | 323 | 182 | 825 |

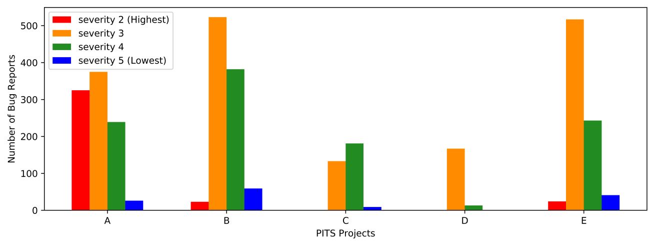

**Figure 2: Distribution of bug report severities in NASA PITS projects**

## 3.3 Classifier

Unlike past studies, we have not tied a specific classifier to our method. We have done this because we believe the classifier should be chosen for the specific use case and domain. With respect to bug reports, many machine learning algorithms have been shown to work well [2]. For our evaluation, we used the multi-layer perceptron because the performance of the algorithm was consistent in previous studies and because the algorithm is general in nature [2].

## 4 EVALUATION

### 4.1 Dataset

For our experiments we used bug reports submitted for five NASA robotic mission projects in the PITS dataset [7]. An example of a PITS Bug Report is presented in Table 1. Each PITS project is identified by a letter ID, from A to E, and the bug report severity ratings are from 1 (most severe) to 5 (least severe)[4] (see Figure 2 for more details). We have removed reports that do not have an assigned severity rating because we would not be able to validate our predictions. After removing all unusable reports we are left with 4026 bug reports (see Table 2). For the usable bug report, we analyzed the bug report summaries (labelled *Subject* in the dataset).

### 4.2 Experiment Methodology

Our experiment was designed to test the validity of our bug severity prediction method using past project data to predict severity in a new project. To ensure that we are using the best training data, we ran our experiment with all possible combinations of past projects. For example, we considered the following training data when predicting bug severity in pitsA:

- single past projects (e.g., pitsB)
- pairs of past projects (e.g., pitsB & pitsC)
- triples of past projects (e.g., pitsB & pitsC & pitsD)
- quadruple of past projects (pitsB & pitsC & pitsD & pitsE)

For each combination of training and testing data we followed the approach outlined in Section 3. In addition to evaluating our own method, we also reproduced the SEVERIS approach and evaluation using the methodology described in the original paper [8].

[4]Severity 1 is not included in any of our experiments because it does not occur in the dataset

**Table 3: Top performing training data for each NASA PITS project**

| Test Project | Training Projects |
|---|---|
| pitsA | pits{B, E} |
| pitsB | pits{C, E} |
| pitsC | pits{A, B} |
| pitsD | pits{B, C, E} |
| pitsE | pits{A, B} |

**Table 4: Precision, recall and f-measure of our prediction method for each NASA PITS project using the training data in Table 3**

| Project | Precision | Recall | F-Measure |
|---|---|---|---|
| pitsA | 0.28 | 0.41 | 0.33 |
| pitsB | 0.53 | 0.54 | 0.53 |
| pitsC | 0.86 | 0.81 | 0.84 |
| pitsD | 0.93 | 0.88 | 0.91 |
| pitsE | 0.55 | 0.57 | 0.56 |

### 4.3 Performance Measures

For each experiment we calculated the precision, recall and weighted f-measure to measure performance. Precision is a score from 1.0 to 0.0 where 1.0 means that all instances that were predicted to be of severity 3 are in fact severity 3. However, precision does not measure the number of instances where the are severity 3 but were predicted to be another severity. Recall is then used to measure this and is also a score between 1.0 and 0.0. F-measure is the weighted harmonic mean of the precision and recall [8]. We present the best f-measure results from our experiments for each project as our results (see Table 4). The projects that were used as training data to produce the results can be found in Table 3.

### 4.4 Discussion

Our results in Table 4 confirm that pitsC and pitsD performed very well with f-measures of 0.84 and 0.91 respectfully. Futhermore, pitsC, pitsD and pitsE all outperformed the SEVERIS approach with respect to f-measure (see Figure 3). These results in particular are surprising when you consider that our method's training data did

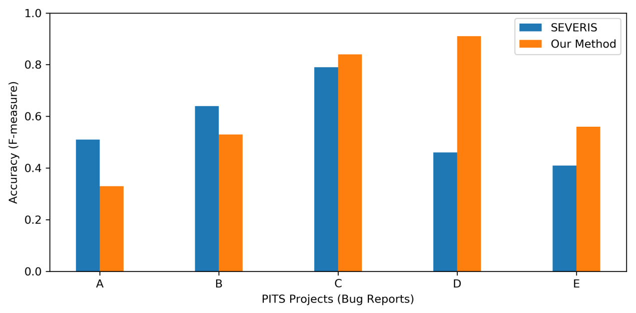

**Figure 3: Comparing the f-measure of the SEVERIS tool with our method on the NASA PITS dataset**

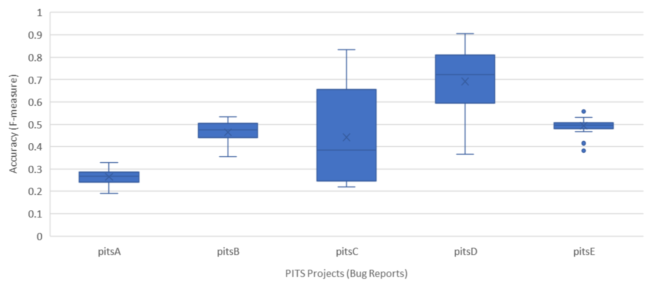

**Figure 4: Accuracy of our method on the NASA PITS dataset with all combinations of training data for each project**

not contain project-specific keyword data. *From these results we are able to confirm that our method is both an effective bug severity predictor and that the severity of a bug can indeed be predicted at the beginning of a project using past project training data.*

The results were not all positive and a further review of the data in Table 4 shows that pitsA was especially poor performing with an f-measure of 0.33. The distribution of bug severities in our dataset (see Figure 2) provides insight as to why this may be the case – almost all instances of severity 2 occur in pitsA (325 bugs) and are sparse across the other projects (48 bugs combined). This result is likely a case of an insufficient number of training samples. Figure 4 shows that for pitsC and pitsD there are varying accuracy results for different combinations of past project training data. *From these results we are able to confirm that our method using past project data is only viable when the training data is representative of the test data.*

The primary threat to the validity of our results is generalizability. We have only evaluated our prediction method on the NASA PITS dataset. However, we believe our method's use of generic document features, the extensive number of experiments conducted and the size of the PITS dataset provides confidence that our method has merit in a more general context.

## 5 CONCLUSIONS

The assignment of severity ratings to bug reports requires both developer time and expertise. While a number of automatic and semi-automatic methods exist to predict bug severity ratings, these methods often need to be trained on bug reports from the current project in order to work effectively and thus are not able to perform predictions until the later stages of development. We have presented a new method to use characteristics of how past projects' bug reports are written to predict the severity of bugs found at the beginning of a new project. Furthermore, our method does not rely on any project- or domain-specific classifier features. The main limitation of our work is that we are reliant on the existence of historic bug reports and the semantic similarity of these bug reports with those of new projects.

There are three avenues of future work we plan to pursue:

(1) Evaluation of our method on additional open source datasets (e.g., the Mozilla dataset) including cross-organizational experiments within a given domain.
(2) Investigation of techniques to improve the selection of training data and exploration of the use of project data subsets for training. For example, a potential approach maybe to find the best training data for each bug severity instead of using the data from an entire past project.
(3) Development of an approach to the cold start problem of selecting what projects should be used as training data when there is little or no information on the current project.

## ACKNOWLEDGMENTS

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference number 2018-06588].

## REFERENCES

[1] John Anvik, Lyndon Hiew, and Gail C Murphy. 2006. Who should fix this bug?. In *Proc. of the 28th Int. Conf. on Soft. Eng. (ICSE 2006)*. 361–370.
[2] Rajni Jindal, Ruchika Malhotra, and Abha Jain. 2017. Prediction of defect severity by mining software project reports. *Int. J. of System Assurance Engineering and Management* 8, 2 (2017), 334–351.
[3] Ahmed Lamkanfi, Serge Demeyer, Emanuel Giger, and Bart Goethals. 2010. Predicting the severity of a reported bug. In *Proc. of the 7th IEEE Work. Conf. on Mining Software Repositories (MSR 2010)*. 1–10.
[4] Ahmed Lamkanfi, Serge Demeyer, Quinten David Soetens, and Tim Verdonck. 2011. Comparing mining algorithms for predicting the severity of a reported bug. In *Proc. of the 15th European Conf. on Software Maintenance and Reengineering (CSMR 2011)*. 249–258.
[5] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proc. of the 31st Int. Conf. on Machine Learning (ICML 2014)*. 1188–1196.
[6] C. Liu, J. Yang, L. Tan, and M. Hafiz. 2013. R2Fix: automatically generating bug fixes from bug reports. In *Proc. of the 6th Int. Conf. on Software Testing, Verification and Validation (ICST 2013)*. 282–291.
[7] Tim Menzies. 2008. *pitsA, pitsB, pitsC, pitsD, pitsE [dataset]*. https://doi.org/10.5281/zenodo.{268475,439580,439581,268447,439582,268513}
[8] Tim Menzies and Andrian Marcus. 2008. Automated severity assessment of software defect reports. In *Proc. of the 24th IEEE Int. Conf. on Software Maintenance (ICSM 2008)*. 346–355.
[9] Xin Rong. 2014. word2vec Parameter Learning Explained. *arXiv preprint arXiv:1411.2738 [cs.CL]* (2014).
[10] Ferdian Thung, David Lo, and Lingxiao Jiang. 2012. Automatic defect categorization. In *Proc. of the 19th Work. Conf. on Reverse Eng. (WCRE 2012)*. 205–214.
[11] Tao Zhang, Jiachi Chen, Geunseok Yang, Byungjeong Lee, and Xiapu Luo. 2016. Towards more accurate severity prediction and fixer recommendation of software bugs. *J. of Systems and Software* 117 (2016), 166–184.
[12] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. 2009. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proc. of the 7th Joint Meeting of the European Soft. Eng. Conf. and the ACM SIGSOFT Symp. on The Foundations of Soft. Eng. (ESEC/FSE 2009)*. 91–100.