Dear Editors and Reviewers,

Thank you for the detailed and constructive reviews.

Following your comments and advice, we have significantly revised our manuscript. Below we provide our response to the comments. We are happy to answer any further questions.

We would like to thank you for giving us this opportunity to improve the manuscript. We look forward to hearing from you.

Yours Sincerely,

Duy, Ferdian, and David

==============================================================================

**Responses to Editor and Reviewers’ comments:**

**Editor**: “Thank you for your submission to the Empirical Software Engineering Journal.

All reviewers see the potential in the work, but they all pointed major concerns that need to be addressed. Among others, we believe it is crucial to address the following main concerns: 1) significance and practicality of the problem, 2) soundness of the techniques (e.g., use of topic modelling), 3) reproducibility of the work (e.g., the data collection approach) and 4) novelty and contributions compared to the conference version.”

**Response**: Thank you for the constructive advice and comments. We have addressed all comments and advice that we have received. The following is our categorization of our responses to reviewers’ comments to address the main concerns:

1. Significance and practicality of the problem

* Clarification on why our proposed tool is needed (Comment 2.1, 2.3, 2.5, 2.6)
* Related work in query performance prediction in information retrieval (Comment 2.13)
* How our approach can be used by developers (Comment 2.6)

1. Soundness of the techniques (e.g., use of topic modeling):

* Usage of LDA model (Comment 1.3, 3.9)
* Clarification on Le and Lo’s approach, i.e., the baseline (Comment 1.5)
* Clarification of RQ1’s results, i.e., distribution of precision, recall, F-measure for all iterations of the ten-fold cross-validation, confusion matrix, precision values (Comment 1.6, 3.11)
* Cross-project setting (Comment 1.7, 3.4, 3.12)
* Clarification on whether Algorithm 1’s output is a classifier or a set of classifiers (Comment 1.1)
* Clarification on Camel case splitting (Comment 2.9)
* Rationale behind severity/priority of bug reports used as metadata features (Comment 2.16)
* Clarification on the learning process of APRILE+ (Comment 2.18)
* Clarification on how our approach helps users/developers by reducing useless bug localization outputs (Comment 2.25)
* Discussion on important features (Comment 2.26, 3.4, 3.12)
* Updates on threats to validity (Comment 2.29, 3.13)
* Usage of multiple SVM classifiers in APRILE (Comment 3.2)
* Applying feature selection to APRILE+ (Comment 3.3)
* Tuning coefficients of component models in APRILE (Comment 3.10)

1. Reproducibility of the work (e.g., the data collection approach):

* How dataset is collected (Comment 1.4)
* Dataset’s download link (Comment 1.9)
* How to obtain buggy source code corresponding to bug reports (Comment 2.21)

1. Novelty and contributions compared to the conference version

* Use of expanded dataset and additional experiments considering the cross-project setting (Comment 1.7, 3.5)
* Clarification on the contribution and novelty points of the paper (Comment 2.8, 3.1)

**Reviewer #1:**

* 1. **Comment**: “(Q1) Mixing Aprile and Aprile+ is confusing. I recommend to only present Aprile+ which is indeed the main novelty of the paper, and hence to merge Section 3 and 4.”

**Response**: Thank you for the advice. We merged Section 3 and Section 4 in the previous submission into Section 3 in the current revision. At first, we describe about features that we extract from IR-based bug localization instances in Section 3.1. Next, Section 3.2 discusses technical aspects of APRILE. Then, we describe APRILE+ in Section 3.3

* 1. **Comment**: “(Q2) The concept of bagging should be presented in Section 2.”

**Response**: Thank you for the advice. We included Section 2.2 (Bagging) in the current submission to address the comment as follows:

“Bagging (Breiman, 1996b), which stands for bootstrap aggregation, is a method to improve accuracy and stability of classification models (i.e., classifiers). Despite of its simplicity, bagging is an effective method for ensemble learning (i.e., combining multiple classification models) as it reduces variance and avoids overfitting. Originally, the method is proposed to work with tree-based models, but it is applicable to other classification models. Given a training data, bagging method creates several samples (with replacement). Each sample is referred to as a bootstrap sample, and used for constructing a base classification model. Subsequently, the output of these models are combined by voting to return one final output.”

(Breiman, 1996b) Breiman L Bagging predictors. Machine Learning 24:123–140

* 1. **Comment**: “(Q3) To me it's a problem to use the topic model data as features. While it works in the context of cross-validation, when a new bug report comes, it does not come with topic model features. The latter case is the standard usage of your technique. The only possibility would be to run a new topic modeling phase for each new bug report, but it's very costly, and it would require to match topics from different runs together.”

**Response**: Thank you for the advice. The constructed topic model is capable of generating topic distributions of a document regardless it belongs to or does not belong to the training corpus. Thus, in the deployment phase, every time a new bug report emerges, we apply the trained LDA topic model in the training phase to estimate topic features for that bug report *without re-creating a brand new topic model on the current corpus and the new bug report*.

Admittedly, new bug reports in deployment phase might have hidden topics that one single topic model can miss. That is the reason why we construct several topic models in the training phase to maximize the chance to capture hidden topics in new bug reports.

We revised Section 3.1.3 of this revision to address the comment as follows:

“… Importantly, LDA models can be used to estimate topic probabilities of an unseen bug report that does not belong to the training corpus (Blei et al, 2003). Therefore, in deployment phase, we calculate topic features (i.e., topic probabilities) of emerging bug reports without updating M$\_{5}$, M$\_{10}$, and M$\_{15}$ models. However, new bug reports might have hidden topics that one single topic model cannot capture. For that reason, we construct a number of different topic models (i.e., M$\_{5}$, M$\_{10}$, and M$\_{15}$) in training phase to maximize the coverage on hidden topics of new bug reports …”

(Blei et al, 2003) Blei DM, Ng AY, Jordan MI Latent dirichlet allocation. J Mach Learn Res 3:993–1022

* 1. **Comment**: “The dataset comes from previous work of the authors. However, I see neither in the referred paper nor in this paper how it was obtained. (Q3) How was the ground truth obtained? Are the bugs sampled (how?) or cherry-picked?”

**Response**: Thank you for the advice. Our AspectJ, Eclipse, and SWT dataset are introduced by Zhou et al. to evaluate BugLocator. According to Zhou et al., information of ground truth is obtained by adopting heuristics proposed by Bachmann and Bernstein [1] to link bug reports to source code files. Each project in the dataset has a study period, and *all* bug reports of fixed bugs in the study period are collected and used for evaluation.

We updated Section 4.1 in the current submission to describe how we collect the bugs and ground truth:

“… Originally, the dataset is introduced by Zhou et al. to evaluate BugLocator. Later, Saha et al (2013) and Wang and Lo (2014) also utilize these bug reports to evaluate BLUiR and AmaLgam. According to Zhou et al (2012), for each software project, **all** bug reports of fixed bugs in the study periods shown in Table 5 are collected. To find the ground truth (i.e., faulty files), BugLocator's authors adopt heuristics proposed by Bachmann and Bernstein (2009) to link bug reports to source code files … The textual bug reports and ground-truth (i.e., faulty files) of our dataset are publicly available**7**. **”**

(Bachmann and Bernstein (2009)) Bachmann, Adrian, and Abraham Bernstein. "Software process data quality and characteristics: a historical view on open and closed source projects." Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops. ACM, 2009.

7https://goo.gl/4sfoj4

Furthermore, we follow comment 3.5 to expand the dataset by including 341 Apache Tomcat bug reports. To collect these Tomcat bug reports and the corresponding ground truth, we first download bug reports submitted between January 2010 and January 2014. For each bug report, we infer the ground-truth (i.e., faulty files) by locating the corresponding commit(s) that fixes the bug in Tomcat’s repository. Then, we exclude bug reports where the name of faulty files are explicitly mentioned in the summaries and descriptions. For these bug reports, it is unnecessary to run bug localization tools since faulty files are already identified in the reports – see (Kochhar et al., 2014). We describe how we collect Tomcat bugs in Section 4.1 as follows:

“… Furthermore, we extend our original dataset by manually collecting issue reports and source code files from Apache Tomcat. We exclude bug reports for which the names of faulty files are explicitly mentioned in the summaries and descriptions of the bug reports. For these bug reports, it is unnecessary to run bug localization tools (Kochhar et al., 2014). Therefore, it is also unnecessary to use our approach to predict the effectiveness of bug localization instances of these bugs....”

(Kochhar et al., 2014) [Pavneet Singh Kochhar](http://dblp.uni-trier.de/pers/hc/k/Kochhar:Pavneet_Singh), [Yuan Tian](http://dblp.uni-trier.de/pers/hc/t/Tian:Yuan), David Lo: Potential biases in bug localization: do they matter? [ASE 2014](http://dblp.uni-trier.de/db/conf/kbse/ase2014.html#KochharTL14): 803-814

* 1. **Comment**: “Q(4) You use [Le and Lo] as baseline, while it is designed for a rather different problem. As you say, there is no trace in bug localization. I don't understand how you can use this as a baseline since it takes completely different data as input. More explanation is needed there.”

**Response**: Thank you for the advice.

Le and Lo’s approach extracts features from execution traces and suspiciousness scores. We adapt Le and Lo’s approach to work for IR-based bug localization by only extracting features from suspiciousness scores. We modify Le and Lo’s approach to not extract features from execution traces since there are no execution traces for many bug reports in the IR-based bug localization setting.

We believe that the adapted version of Le and Lo’s approach is a good baseline to compare with APRILE+. In fact, Le and Lo’s approach is the first one to predict the effectiveness of an automated debugging technique.

We update Section 4.2 with our explanation as follows:

“Recently, Le and Lo propose an approach to predict the effectiveness of a spectrum-based bug localization tool (Le and Lo, 2013; Le et al, 2014a). A spectrum-based bug localization tool analyzes a set of failed and correct execution traces, and computes suspiciousness scores of program elements (e.g., statements). *Le and Lo's approach is the first study that predicts the effectiveness of automated debugging tools, and thus it is closely related to our work. Le and Lo's approach uses features extracted from program execution traces and suspiciousness scores. In IR-based bug localization setting, there is no execution traces (since most bug reports do not come with execution traces); thus we can only run Le and Lo's approach on features that are extracted from suspiciousness scores of files.* We use this approach as a baseline to compare with APRILE+. We denote this baseline as SVMExt,Score. Furthermore, we also compare APRILE+ against APRILE.”

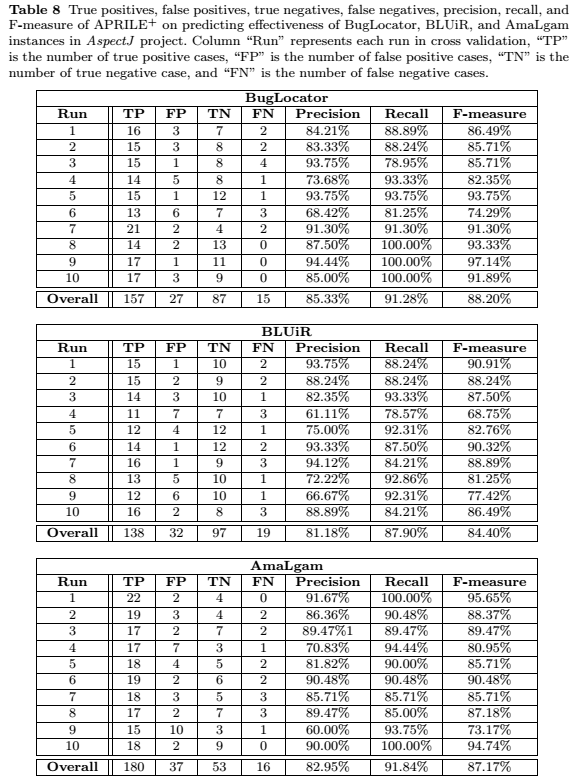
* 1. **Comment**: “Table 7: Since K=1, the precision is either one or zero. If the number of files to be retrieved is one, the recall is also one or zero. (Q5) I would like to see the distribution of precision, recall and F-score for alls run of cross-validation (i.e. for 3459 predictions). Then, for Eclipse you say that the average precision is 59%, it means that you have 3075\*.59=1814 perfect predictions. However, you say that you have 754+1594=2348 perfect predictions. (Q6) This does not correspond. Why? (Q7) I'd like to see all absolute numbers of the confusion matrix in a table as well.”

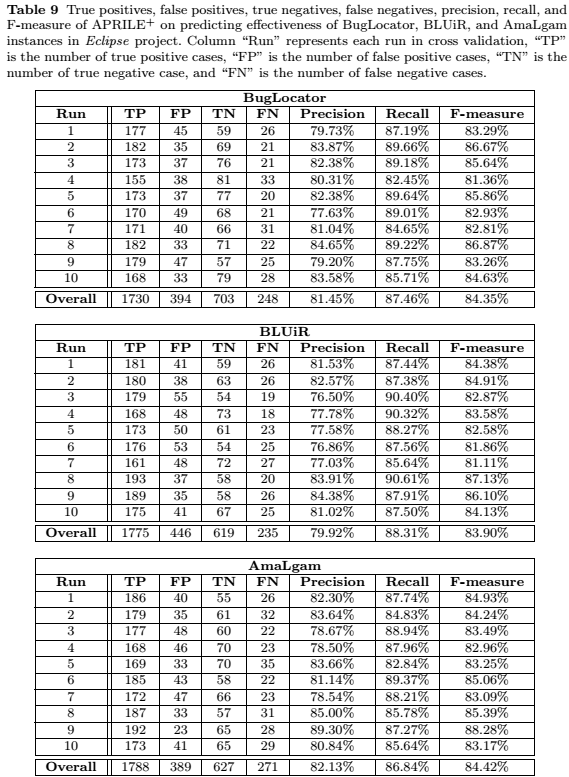
**Response**: Thank you for the advice. There seems to be a misunderstanding. In Section 4.1, we define precision as the ratio between the number of true positives (i.e., effective bug localization instances are correctly predicted) and the total number of instances that are predicted as effective. Note that the definition of precision (or recall) for classification problem is different from the definition of precision (or recall) in information retrieval. Therefore, when N=1, precision possibly ranges between 0% and 100% and is not limited to only either one or zero.

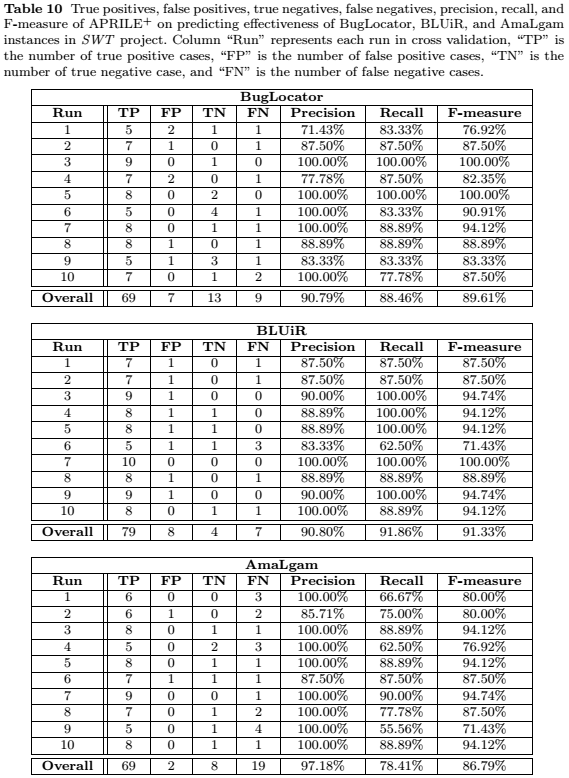
Table 7 in the previous revision shows that the precision of APRILE+ for Eclipse is 59.79%. That means among (754+507) instances that are predicted as effective there are 59.79% x (754+507) = 754 instances that are actually effective. Note 754 is the number of true positive cases, and 507 is the number of false positive cases, and 1,594 is the number of true negatives cases.

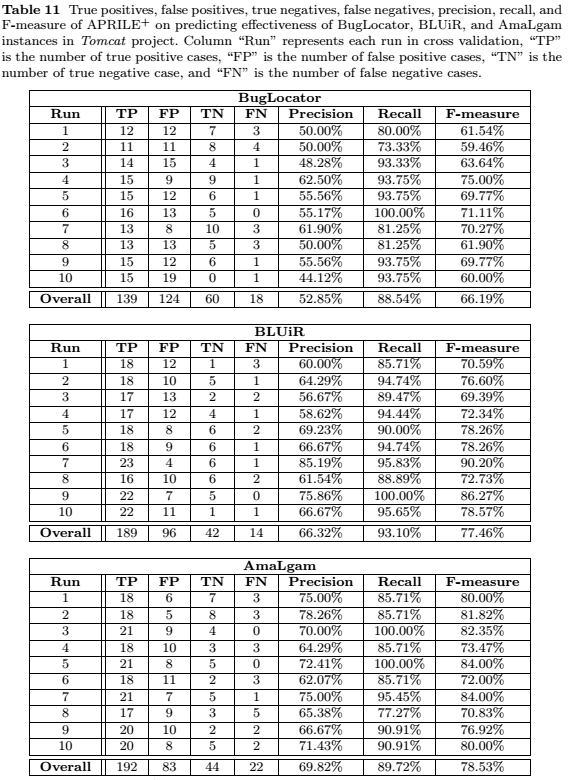
Note that in this revision, we merge RQ5 into RQ1, and change the default effectiveness criterion to N=10 (see Comment 2.22, 2.24, 2.27, and 2.28 for further details) as well as include the new Tomcat dataset (see Comment 3.5). Thus, there are changes in the results of APRILE+ on BugLocator instances.

Following your advice, in the current revision (Tables 8, 9, 10), we have included additional statistics of APRILE+‘s predictions in each cross-validation run. The included statistics are the number of true positives, false positives, true negatives, false negatives, precision, recall, and F-measure. Tables 8, 9, 10, 11 are shown below:









* 1. **Comment**: “(Q8) In the features, some are project-specific and others and generic. If you only consider the generic ones, could you train the system on a project and evaluate it on another one? It would be great for applying such a system on new projects with no bug data.”

**Response**: Thank you for the comment. We have included research question 5 to evaluate APRILE+ in cross-project setting. According to the empirical result, we find that precision, recall, and F-measure of APRILE+ in cross-project setting are lower than those in standard cross-validation setting. This is as expected since cross-project prediction is much harder than within-project prediction – c.f., (Zimmermann et al., 2009).

(Zimmermann et al., 2009) Thomas Zimmermann, [Nachiappan Nagappan](http://dblp.uni-trier.de/pers/hc/n/Nagappan:Nachiappan), [Harald C. Gall](http://dblp.uni-trier.de/pers/hc/g/Gall:Harald_C=), [Emanuel Giger](http://dblp.uni-trier.de/pers/hc/g/Giger:Emanuel), [Brendan Murphy](http://dblp.uni-trier.de/pers/hc/m/Murphy:Brendan): Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. [ESEC/SIGSOFT FSE 2009](http://dblp.uni-trier.de/db/conf/sigsoft/fse2009.html#ZimmermannNGGM09): 91-100

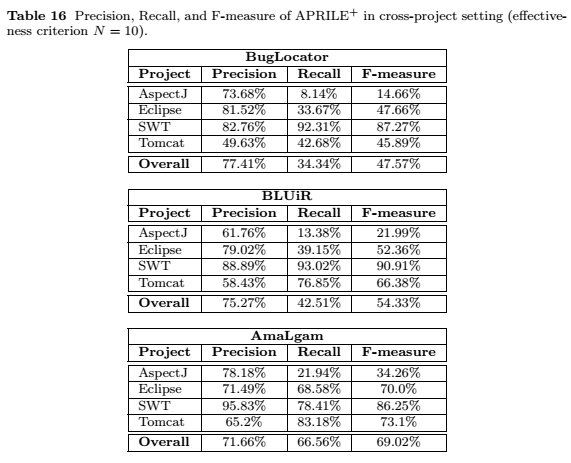
The research question 5 is described as follows:

“RQ5: Could bug localization instances from one software project be used to learn a model for predicting effectiveness of instances of another software project?

In new software projects, the amount of bug localization instances are not always sufficient to formulate good training data. Therefore, we investigate the effectiveness of APRILE+ in cross-project setting in this research question. Assuming we have bug localization instances from P different software project. We learn APRILE+'s prediction model from instances of P-1 projects. This model is then employed to predict effectiveness of bug localization instances from the other project.”

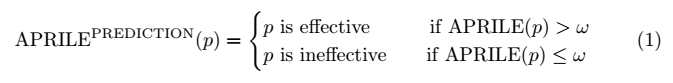
And the following text describes the result which answers research question 5:

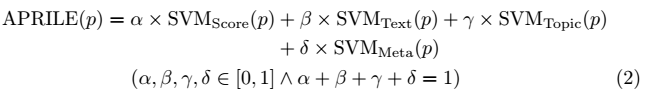
“To answer this research question, we apply APRILE+ to predict the effectiveness of BugLocator, BLUiR, and AmaLgam instances with effectiveness criteria N = 10 in cross-project setting. Table 16 shows precision, recall, and F-measure of APRILE+ among the investigated bug localization tools. According to the table, APRILE+ achieves the overall F-measure of 47.57%, 54.33%, and 69.02% when predicting effectiveness of BugLocator, BLUiR, and AmaLgam instances, respectively. Noticeably, compared to Table 7 (i.e., standard cross-validation setting), we find F-measure of APRILE+ significantly reduces in most of projects and bug localization tools. This is as expected since cross-project prediction is much harder than within-project prediction – c.f., (Zimmermann et al., 2009). However, F-measures of APRILE+ remain high (up to 90.91%) in SWT for all instances of BugLocator, BLUiR, and AmaLgam. This implies that it is still possibly to use bug localization instances of one or many projects to infer APRILE+’s models for predicting effectiveness of instances in another project.”



* 1. **Comment**: “I recommend to introduce omega in Equation 1, I expected its definition there.”

**Response**: Thank you for the advice. Equation 1 in the previous submission becomes Equation 2 in the current submission. We included the information of omega into Equation 2 as follows:





* 1. **Comment**: “Beginning of 5.1 - dataset - add a link to the dataset.”

**Response**: Thank you for the advice. We have included the download link of the dataset of the four projects (i.e., https://goo.gl/4sfoj4) in Section 4.1 of the current revision. Please refer to our response to comment 1.4 for the revised text of Section 4.1.

* 1. **Comment**: “SVN\_{Score}^{Ext} is a rather unintuitive acronym. Avoid both sub- and superscripts? Same for CLA\_{FC}^{BEC}”

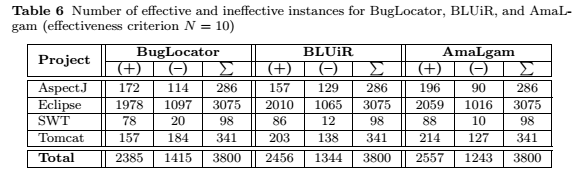
**Response**: Thank you for the advice. We changed our acronyms by moving superscripts to subscripts. For example, SVMExt,Score, CLABEC,FC, etc.

* 1. **Comment**: “Algo 1: the output is simply a set of classifiers, it's not a classifier per se?”

**Response**: Thank you for the advice. The output of Algorithm 1 is a classifier which is trained by a bagging based ensemble algorithm. It is a composite classifier that is a composition of a number of classifiers.

* 1. **Comment**: “Table 6; sums over rows would be useful.”

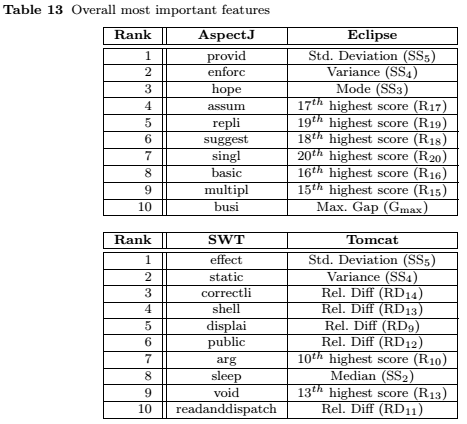
**Response**: Thank you for the advice. Table 6 is updated as follows:



Note that the table has also been updated (e.g., by adding a new row for Tomcat) to address other comments (please refer to comment 2.22, 2.24, 2.27, 2.28, and 3.5 for further details).

* 1. **Comment**: “Table 9: give human readable labels for features. (what is MT\_6? etc.)”

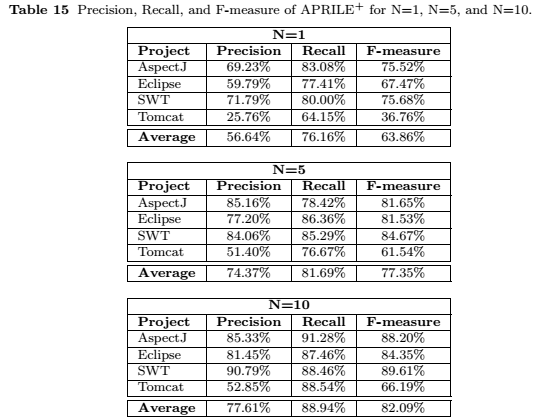
Response: Thank you for the advice. We have included human readable labels of the best features in Table 9 (i.e., Table 13 in the current revision) as follows:

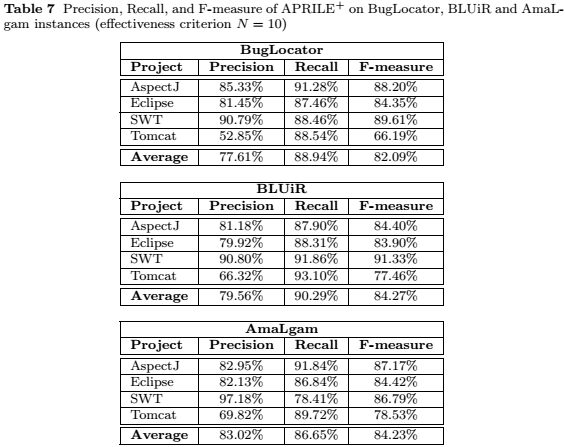


Note that we follow Comment 2.26 to show top-10 most important features across the four categories of features. We also include the top-10 most important features from the additional Tomcat dataset.

* 1. **Comment:** “Table 11 and 13 give also precision and recall as Table 7 and Table 8”

**Response:** Thank you for the advice. In the current revision, we have also included values of precision and recall in Tables 11 and 13 (i.e., Tables 15 and 7 in the current revision, respectively) as follows:





Note that in this revision, we merge RQ5 into RQ1 and change effectiveness criterion to N=10. We also include the additional Tomcat dataset. Please refer to comment 2.22, 2.24, 2.27, 2.28, and 3.5 for further details.

**Reviewer #2:**

1. **Comment**: “Summary of Opinion: The only issue I have with this work is that I'm not convinced that it is attacking an important problem. I believe their approach is good, the results are strong, and the validation was thorough. *However, is the meta-tool needed? I'm not sure.* I'd recommend for acceptance with (relatively minor) revision, as I think they need to make a stronger **case** for *why this tool is needed in the Abstract and Intro.”*

**Response**: Thank you for the advice. We revised the abstract and Introduction section to address comment 2.3, 2.5, 2.6, 2.7. Please kindly proceed to these comments for further details.

1. **Comment**:

“Positives:

\* Clear writing, easy-to-understand

\* Strong experimental work on a large number of bugs from real systems

\* Sound approach

\* Sound extension to the approach

\* Strong results, especially when N = 10”

**Response**: Thank you for the strong support to the work !

1. **Comment**: “Issues: - The problem seems insignificant. Part of this is writing, but part of this is the problem itself. *Do we need a meta-tool to tell use when to trust the original tool?* *Why not just try to improve the original tool?”*

**Response**: Thank you for the questions.

We strongly believe that the answer is “yes”. Please kindly consider the following rationales:

* The current state-of-the-art information retrieval (IR) based bug localization tools are still far from perfect. Researchers keep improving IR based bug localization by proposing many new tools that are shown to outperform previously created ones. *However, there is NO perfect bug localization tools that can successfully localize faults within a few number of most suspicious program elements for every single input bug report.* It is unclear whether there will ever be such a tool in the future. These mean that there will be bug reports where a more advanced bug localization tool is bad (i.e., ineffective), but a less advanced one is good at these bug reports. With the many IR based bug localization tools proposed in the literature (e.g., BugLocator, BLUiR, AmaLgam etc.), it is difficult for developers to decide which tool would be effective for a given bug report. Therefore, a meta-approach that can tell developers which bug localization tool is likely to help developers locate the fault for a given bug report would be needed. This need motivates our work.
* Moreover, for some bug reports, no bug localization tools would be useful. Developers would waste time to go through the output of bug localization tools. Developers would also likely to lose confidence on an ``oracle’’ that are ``wrong’’ many times. The problem goes beyond the power of such bug localization tool but also on the quality of the input data (bug report). Thus, there is a need to inform developers that none of the tools are useful for some cases. Developers can then use traditional debugging to find bugs and not waste time with automated debugging. This need also motivates our work.

We have added the following text in the abstract of the paper:

“… Recently, several IR-based bug localization tools have been proposed. However, there are no perfect tools that can successfully localize faults within a few number of most suspicious program elements for every single input bug report. Therefore, it is difficult for developers to decide which tool would be effective for a given bug report. Furthermore, for some bug reports, no bug localization tools would be useful. Even a state-of-the-art bug localization tool outputs many ranked lists where buggy files appear very low in the lists. This potentially causes developers to distrust bug localization tools …”

We have added the following text in the introduction section of the paper:

“… If an IR-based bug localization tool is effective, developers should be able to find a buggy file by inspecting just a few files at the top of the ranked list. Unfortunately, the current state-of-the-art IR-based bug localization tools are imperfect. Currently, there is **no** perfect bug localization tools that can successfully localize faults within a few number of most suspicious program elements for every single input bug report. It is unclear whether there will ever be such a tool in the future. These mean that there will be bug reports where a more advanced bug localization tool is bad (i.e., ineffective), but a less advanced one is good at these bug reports. With the many IR based bug localization tools proposed in the literature, e.g., (Saha et al, 2013; Wang and Lo, 2014; Zhou et al, 2012), etc., it is difficult for developers to decide which tool would be effective for a given bug report. Moreover, for some bug reports, no bug localization tools would be useful. Developers would waste time to go through the output of bug localization tools. Recently, Parnin and Orso conducted a user study on an automatic debugging tool, and find that developers do not find an automatic debugging tool useful if they cannot find the root cause of a bug early in a ranked list (Parnin and Orso, 2011). These ineffective cases can make developers lose confidence in bug localization tools…”

1. **Comment**: “Issues: - Some redundancy across sections... be careful not to repeat yourself.”

**Response**: Thank you for the advice. We have revised the current revision to avoid redundant text. Please refer to our responses to comment 1.1, 2.13, and 2.23 for more details.

1. **Comment**: “Abstract: The abstract takes too long to describe the problem that you are solving. Get to the point faster.

The problem you are attacking comes across as a very small part of a larger problem. If I understand correctly, you have created an oracle that predicts whether a bug localization tool is likely to be effective or not for a given problem. Unfortunately there is little evidence that developers use bug localization tools in the first place, and so this line of work comes across as building castles in the air. (Not sure if that translates... I mean you have done excellent work, but you're *building upon other tools that are not used in practice*, so I worry about the potential impact of this work). Part of that is inherent in the work, and this part you can't fix in the context of this paper. However, part is in the presentation of the work. If the abstract got to the point more quickly it wouldn't be so obvious that this is a small sub-problem of another problem.”

**Response**: Thank you for the advice. As requested, we shorten and update the abstract. The following is the updated abstract:

“Information retrieval (IR) based bug localization approaches process a textual bug report and a collection of source code ﬁles to ﬁnd buggy ﬁles. They output a ranked list of ﬁles sorted by their likelihood to contain the bug. In this work, we build an oracle that can automatically predict whether a ranked list produced by an IR-based bug localization tool is likely to be eﬀective or not. We consider a ranked list to be eﬀective if a buggy ﬁle appears in the top-N position of the list. If a ranked list is unlikely to be eﬀective, developers do not need to waste time in checking the recommended ﬁles one by one. In such cases, it is better for developers to use traditional debugging methods or request for further information to localize bugs. To build this oracle, our approach extracts features that can be divided into four categories: score features, textual features, topic model features, and metadata features. We build a separate prediction model for each category, and combine them to create a composite prediction model which is used as the oracle. We name this solution APRILE, which stands for Automated PRediction of IR-based Bug Localization’s Eﬀectiveness. We further integrate APRILE with two other components that are learned using our bagging-based ensemble classiﬁcation (BEC) method. We refer to the extension of APRILE as APRILE+.

We have evaluated APRILE+ to predict the effectiveness of three state-of-the-art IR-based bug localization tools on more than three thousands bug reports from AspectJ, Eclipse, SWT, and Tomcat. APRILE+ can achieve an average precision, recall, and F-measure of 77.61%, 88.94%, and 82.09%, respectively. Furthermore, APRILE+ outperforms a baseline approach by Le and Lo and APRILE by up to a 17.43% and 10.51% increase in F-measure respectively.”

1. **Comment**: “Introduction: You never explicitly state how your prediction tool would be used in practice. Would it be used by the IDE to help decide whether to show a particular bug localization tool's output? Or simply by the user? Explain how this solution would be used in the context of fixing a bug.”

**Response**: Thank you for the advice. We have described the usage scenario of our approach by adding the following text in the Introduction section (Section 1):

“Our approach can potentially be integrated to an IDE and a bug tracking system. Given a new bug report to be debugged in a bug tracking system (e.g., JIRA or Bugzilla), our approach would run a number of IR based bug localization techniques in the background, and predicts their effectiveness. It will then pick the bug localization tool output that is most likely to be effective, and highlight potentially buggy files to developers in the IDE. If none of the outputs are likely to be successful, it will notify the developer that it has no good recommendation and the developer can either improve the description of the bug report or proceed with traditional debugging. Our approach can also be used standalone. For this setting, developers can directly input the directory containing the source code files, and the text in a bug report that he/she wants to debug, and our approach can process this input to produce a list of potentially buggy files, or declare that no good recommendation can be made.”

1. **Comment**: “Introduction: The results are not clearly stated. I would break he results paragraph into two parts. The first part that stops at "..72.89%, respectively." And the second part that discusses how it outperformed the original approach.”

**Response**: Thank you for the advice. We have divided the paragraph describing the results into 2 smaller ones as suggested.

Furthermore, we follow comment 3.5 to extend the dataset by including new Tomcat dataset. Since the total number of bugs in the dataset, the average precision, recall, and F-measure are changed. We update a part of Introduction as follows:

“…In terms of F-measure (i.e., the harmonic mean of precision and recall), APRILE+ outperforms a baseline based on the approach proposed by Le and Lo (Le and Lo, 2013; Le et al, 2014a) by up to 17.43%. Le and Lo proposed an approach that predicts the effectiveness of a spectrum-based bug localization tool, e.g., Tarantula (Jones and Harrold, 2005). Spectrum-based bug localization tools analyze execution traces rather than bug reports. We adapt their approach to predict the effectiveness of an IR-based bug localization tool and use it as the baseline. We have also compared the performance of APRILE+ against the performance of APRILE and find that APRILE+ can outperform APRILE for all datasets by up to a 10.51% increase in F-measure…”

1. **Comment**: “Introduction: The contributions restate a lot of what was said in the previous paragraph, especially #4. Are both necessary?”

**Response**: Thank you for the advice. We shorten the 4th contribution as follows

“4. We evaluate our approach on a dataset of 3,800 bug reports from three software projects. The empirical results show that APRILE+ performs well for various datasets and settings.”

Note that we follow comment 3.5 to include additional dataset. In total, we evaluate 3,800 bug reports.

1. **Comment**: “Background: Camel case splitting is not the standard when processing programming languages. It would fail on ASTParser, for instance.”

**Response**: Thank you for the advice. In our experiments, we use Java’s regular expression API to split Java identifiers to smaller tokens. We use the following regular expression: "(?<!(^|[A-Z]))(?=[A-Z])|(?<!^)(?=[A-Z][a-z])". This regular expression splits “ASTParser” into “AST” and “Parser”. In the this revision, we describe how we use Java’s regular expresion API to perform identifier splitting in Section 4.1 as follows:

“**Text Preprocessing**: In our experiments, we apply Java's regular expression API to split identifiers to smaller words following the Camel casing convention. We use the following regular expression to split an identifier: "(?<!(^|[A-Z]))(?=[A-Z])|(?<!^)(?=[A-Z][a-z])". This regular expression makes sure a string is split according to Camel casing convention. e.g., ``processFile'' is split to ‘’process’’ and ‘’File’’. It also correctly detects acronyms, e.g., ``ASTParser'' is split to ``AST'' and ``Parser''...”

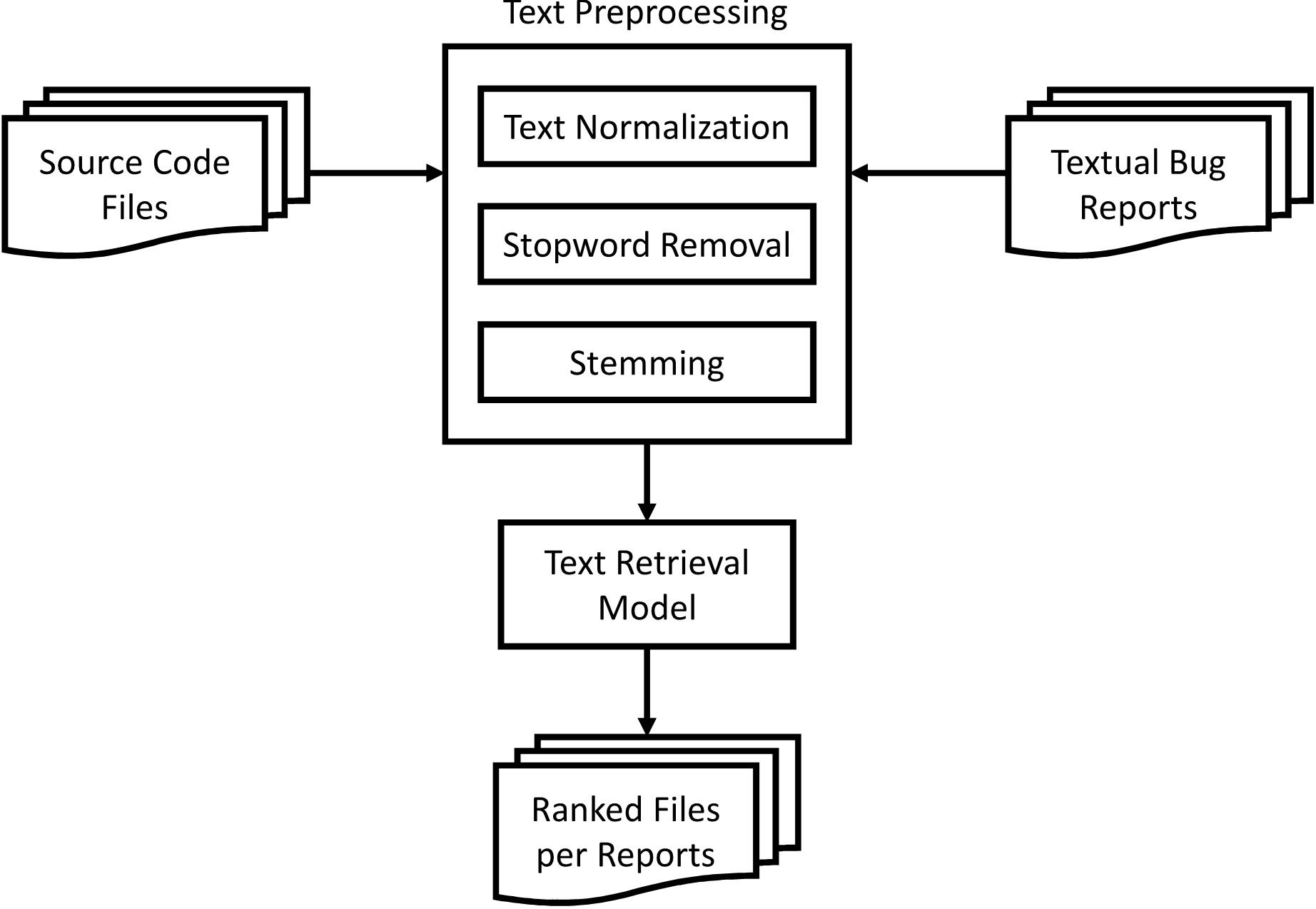
1. **Comment**: “Background: Background information on stemming is mixed in with what you use in your approach (Porter). You should not mention what you use in the background section.”

**Response**: Thank you for the advice. We move the text that mentions Porter Stemming in Section 2.1 of the Background section to Section 4.1 of the Experimental Evaluation section as follows:

“**Text Preprocessing**: … Moreover, we use the Porter Stemming algorithm (Porter, 1980), which is a popular stemming algorithm, to reduce words to their root forms.”

1. **Comment**: “Background: I'd recommend using a diagram to explain the process started on p5 l39.”

**Response**: Thank you for the advice. We created a figure to describe the process of generating a ranked list of source code files for an input bug report as follows



We updated Section 3.1 of the current revision with the following text as follows:

“… Figure 1 shows the process of generating a ranked list of source code files for an input bug report.”

1. **Comment**: “Background: I don't think the background information on SVM is appropriate. I would remove it.”

**Response**: Thank you for the advice. We have removed the section that describes Support Vector Machine in the current revision.

1. **Comment**: “APRILE: So it seems that all of the features listed in Table 1 are related to the "Suspiciousness Score", which is similar the the "similarity score" in web search. Therefore, I imagine that, if your approach is valid, the researchers working on web search might have come up with a similar approach to determining whether their search results are valid. Have you spoken with an expert in the search field to make sure there's not some work you can reuse there?”

**Response**: Thank you for the advice. We have performed more literature survey beyond software engineering and find a line of work on *query performance prediction* in information retrieval domain to be closely related to our study in predicting effectiveness of bug localization tools. That line of research focuses on estimating query difficulty, i.e., the quality of search results retrieved for a query from a given collection of documents. There are two main groups of query performance prediction approaches: pre-retrieval and post-retrieval. In the nutshell, pre-retrieval prediction approaches estimate the quality of query’s results before the retrieval takes place [1,2]. On the hand, post-retrieval prediction approaches analyzes rankings returned by search engines/retrieval systems to estimate quality of query’s results [3,4,5,6]. Our work is closer to post-retrieval approaches based on score distribution analysis [5,6].

In [5,6], Shtok et al. propose NQC (i.e., Normalized Query Commitment) which is a predictor (i.e., measurement) to estimate the query performance. For a given query *q*, the NQC value of *q* is the standard deviation of retrieval scores of all documents normalized by the query likelihood retrieval score of the whole corpus. If NQC of *q* is greater than the mean retrieval score, *q* is estimated as “difficult”. Otherwise, *q* is an “easy” query. Different from Shtok et al.’s, we compute several statistics from suspiciousness scores (i.e., retrieval scores) such as gaps between suspiciousness scores, relative differences, etc., in addition to standard deviation. Furthermore, we extract textual and context-specific features from bug reports besides features from suspiciousness scores. Furthermore, we employ state-of-the-art machine learning classification algorithms to predict the effectiveness of bug localization instances instead of comparing a statistic based on standard deviation of retrieval scores as Shtok et al.’s approach.

[1] Mothe, J. and Tanguy, L., 2005. Linguistic features to predict query difficulty. In ACM Conference on research and Development in Information Retrieval, SIGIR, Predicting query difficulty-methods and applications workshop (pp. 7-10).

Vancouver

[2] He, B. and Ounis, I., 2004, January. Inferring query performance using pre-retrieval predictors. In String processing and information retrieval (pp. 43-54). Springer Berlin Heidelberg.

Vancouver

[3] Vinay, V., Cox, I.J., Milic-Frayling, N. and Wood, K., 2006, August. On ranking the effectiveness of searches. In Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval (pp. 398-404). ACM.

[4] Cronen-Townsend, S., Zhou, Y. and Croft, W.B., 2002, August. Predicting query performance. In Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval (pp. 299-306). ACM.

[5] Shtok, A., Kurland, O. and Carmel, D., 2009. Predicting query performance by query-drift estimation. In Advances in Information Retrieval Theory (pp. 305-312). Springer Berlin Heidelberg.

[6] Shtok, A., Kurland, O., Carmel, D., Raiber, F. and Markovits, G., 2012. Predicting query performance by query-drift estimation. ACM Transactions on Information Systems (TOIS), 30(2), p.11.

We updated the related work section (Section 5) as follows:

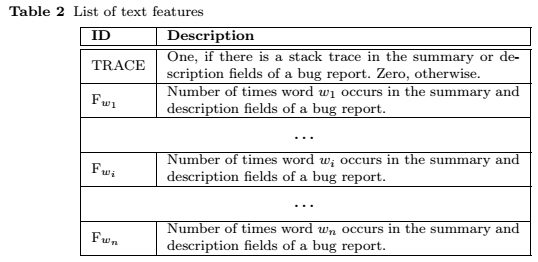
*“*Query Performance Prediction in Information Retrieval

In information retrieval, query performance prediction approaches are the most relevant to our work in predicting effectiveness of bug localization tools. This line of research focuses on estimating query difficulty, i.e., the quality of search results retrieved for a query from a given collection of documents. There are two main groups of query performance prediction approaches: pre-retrieval and post-retrieval. In general, pre-retrieval prediction approaches estimate the quality of query’s results before the retrieval takes place (He and Ounis, 2004; Mothe and Tanguy, 2005). On the hand, post-retrieval prediction approaches analyzes rankings returned by search engines/retrieval systems to estimate quality of query’s results (Cronen-Townsend et al, 2002; Shtok et al, 2009, 2010; Vinay et al, 2006). In comparison with query performance prediction, our work is more relevant to post-retrieval approaches based score distribution analysis (Shtok et al, 2009, 2010).

In (Shtok et al, 2009, 2010), Shtok et al. propose NQC (i.e., Normalized Query Commitment) which is a predictor (i.e., measurement) to estimate the query performance. For a given query q, the NQC value of q is the standard deviation of retrieval scores of all documents normalized by the query likelihood retrieval score of the whole corpus. If NQC of q is greater than the mean retrieval score, q is estimated as “difficult”. Otherwise, q is an “easy” query. Different from Shtok et al.’s, we compute several statistics from suspiciousness scores (i.e., retrieval scores) such as gaps between suspiciousness scores, relative differences, etc., in addition to standard deviation. Furthermore, we extract textual and context-specific features from bug reports besides features from suspiciousness scores. Furthermore, we employ state-of-the-art machine learning classification algorithms to predict the effectiveness of bug localization instances instead of comparing a statistic based on standard deviation of retrieval scores as Shtok et al.’s approach.”

1. **Comment**: “APRILE: I would make it clearer in Table 2 that there are \*many\* F\_w features (one for each word). As is it looks like there are only two features in this category.”

**Response**: Thank you for the advice. We updated Table 2 in this revision as follows:



1. **Comment**: ” APRILE: I'm surprised there wasn't a background section on LDA. Should there be a short one?”

**Response**: Thank you for the advice. In the previous revision, we briefly describe LDA in Section 2.4 (i.e., Topic Modeling). The description is as follows

“Topic modeling is a technique to discover latent topics in a collection of documents. These latent topics are inferred based on the occurrences of words in the documents. One of the most popular topic modeling techniques is Latent Dirichlet Allocation (LDA) (Blei et al, 2003). LDA posits that each document is a mixture of topics and each word in the document is associated to a topic. Given a document, LDA generates its topic distribution, which corresponds to the probability of each topic to be assigned to the document. We apply Latent Dirichlet Allocation (LDA) (Blei et al, 2003) to extract a number of features from bug reports …”

Note that in this revision, we follow comment 3.7 to remove Section 2.4 (i.e., Topic Modeling) and briefly describe topic modeling and LDA in the beginning of Section 3.1.3 (i.e., topic model features).

(Blei et al, 2003) Blei DM, Ng AY, Jordan MI Latent dirichlet allocation. J Mach Learn Res 3:993–1022

1. **Comment**: “APRILE: It is very difficult to understand why priority would have an effect on bug localization. Can you explain why priority and severity, or really any of these features were included? Features in machine learning should usually have some intuition behind them.”

**Response**: Thank you for the question. We updated Section 3.1.4 (Metadata Features) with the intuition of including these features as follows:

Bug reports assigned with high severity or priority are likely to be highly-noticeable bugs and are well described by reporters. Well described bugs often have in their summaries and descriptions important and highly relevant words that lead IR-based bug localization tools to the exact faulty files. Therefore, it is possible that it is easier for IR-based bug localization tools to localize faults of bug reports with high severity/priority.

*“*Table 4 shows a list of metadata features that we are interested in. In total, there are 14 metadata features. Among the features, features MT1 and MT2 capture severity and priority of the reported bug. *Intuitively, bug reports assigned with high severity or priority are typically highly-noticeable bugs that affect major functionalities of an application. These bugs are often well described by reporters, and thus the bug reports are likely to contain important and highly relevant words that can better lead IR-based bug localization tools to the exact faulty files …”*

1. **Comment**: “APRILE+ "goodness" is too informal”

**Response**: Thank you for the advice. We changed the following phase “The idea is to keep the goodness of APRILE and improve upon it” to “The idea is to keep the best features of APRILE and improve upon it” in Section 3.3.1 of the current revision.

1. **Comment**: “APRILE+ : So, to summarize APRILE+, you used two additional machine learning approaches on the same features and then had them use majority voting to make the final decision? If so, then what is the learning?”

**Response**: Thank you for the comment. For APRILE+, the learning stage (i.e., training stage) happens in each of its three components (i.e., APRILE, BEC^SVM, and BEC^RBF). Each component has its own learning phase where features are extracted from training bug localization instances and classification models are constructed according to each approach’s strategy. We have added the following text into Section 3.3.1 of the current revision:

*“…* In the training phase, APRILE+ takes as input a set of training bug localization instances and their corresponding effectiveness labels. These inputs are then used to construct the three classifiers inside APRILE+ *(i.e., APRILE, SVM$^{BEC}$, and RBF$^{BEC}$). Each of these classifiers has its own training phase where features are first extracted from the inputs (see Section 3.1), and prediction models are constructed according its learning strategy (see Section 3.2.2 and Section 3.3.2)…”*

1. **Comment**: “APRILE+: … I'm not a machine learning expert, but doesn't a combination of several learnings and a voting scheme generally outperform a single learner?”

**Response:**  Thank you for the comment. Yes, a combination of several learning algorithms and a voting scheme generally outperforms a single learner [1,2,3,4]. This is the motivation behind the design of APRILE+. We have added the following text in the Introduction section:

*“…* In this journal paper, we propose an extension of APRILE and call it APRILE*+. Intuitively, a combination of several prediction approaches and a voting scheme generally outperforms a single learner (Bauer and Kohavi, 1999; Breiman, 1996a; Lemmens and Croux, 2006; Prasad et al, 2006). Therefore,* APRILE+ extends APRILE by integrating APRILE with two other components that are learned using our bagging-based ensemble classification (BEC) method. Each of these three components will output a recommendation and APRILE+ will output a final recommendation based on majority voting, i.e., the output (effective or ineffective) that is recommended by at least two out of the three components will be the final prediction output...*”*

[1] Bauer, E. and Kohavi, R., 1999. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. Machine learning, 36(1-2), pp.105-139.

[2] Breiman, L., 1996. Bagging predictors. Machine learning, 24(2), pp.123-140.

[3] Lemmens, A. and Croux, C., 2006. Bagging and boosting classification trees to predict churn. Journal of Marketing Research, 43(2), pp.276-286.

[4] Newer classification and regression tree techniques: bagging and random forests for ecological prediction

1. **Comment**: “APRILE+: Fig 2 is very clear and helpful”

**Response**: Thank you for the comment.

1. **Comment**: “Experimental Evaluation: For your dataset... where did you get the code corresponding to the bug? Was it from the main branch, the dev, branch? What is Eclipse's branching scheme? I need more detail than Table 5 in order to replicate you experiment.”

**Response**: Thank you for the comment. We use source code from the stable releases for Eclipse and SWT bugs. For AspectJ bugs, we use the iBugs dataset where each bug has its own faulty source code version. For replication purpose, we have updated Section 4.4.1 with the download links of the source code corresponding to the AspectJ, Eclipse, and SWT bugs that we consider in this work:

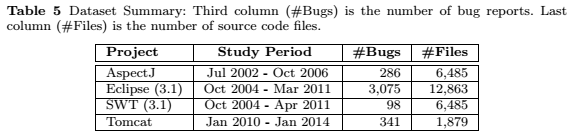
“We conduct experiments using three sets of bug reports and source code files from AspectJ1, Eclipse2, and SWT3 which contain a total of more than three thousands bug reports … ”

3<https://www.st.cs.uni-saarland.de/ibugs/>

4<http://goo.gl/Ojqrrp>

5<https://bugcenter.googlecode.com/files/swt-3.1.zip>

Following comment 3.5, we add 341 Tomcat bugs into the dataset. Textual bug reports and ground-truth of the original dataset (i.e., Eclipse, AspectJ, and SWT) and Tomcat dataset can be downloaded from https://goo.gl/4sfoj4. We also include information of the Tomcat dataset in Section 4.1 as follows:



“…Furthermore, we extend our original dataset by manually collecting issue reports and source code files from Apache Tomcat6. We exclude bug reports for which names of faulty files are explicitly mentioned in the summaries and descriptions of the bug reports. For these bugs, it is unnecessary to run bug localization tools (Kochhar et al, 2014). Therefore, it is also unnecessary to use our approach to predict the effectiveness of bug localization instances of these bugs. Details of the dataset are shown in Table 5. In total, we investigate a dataset of 3,800 bug reports. The textual bug reports and ground-truth (i.e., faulty files) of our dataset are publicly available7.”

6<http://svn.apache.org/repos/asf/tomcat/trunk/>

7https://goo.gl/4sfoj4

Kochhar PS, Tian Y, Lo D (2014) Potential biases in bug localization: do they matter? In: ACM/IEEE International Conference on Automated Software Engineering, ASE ’14, Vasteras, Sweden - September 15 - 19, 2014, pp 803– 814

1. **Comment**: “5.2: For RQ1, why did you only compare against BugLocator? Can you explain why you didn't use the other tools?”

**Response**: Thank you for the advice. We have also used other tools. The experiments with other tools were originally included as part of RQ5. In this revision, we merge RQ5 into RQ1. Please refer to our response to Comment 2.24 for further details.

1. **Comment**: “5.2: RQ2, the explanation of Le and Lo's approach seems to repeat a lot from related work. Shorten.”

Response: Thank you for the advice. We shorten the description of RQ2 as follows:

“Recently, Le and Lo propose an approach to predict the effectiveness of a spectrum-based bug localization tool (Le and Lo, 2013; Le et al. 2014a). A spectrum-based bug localization tool analyzes a set of failed and correct execution traces, and computes suspiciousness scores of program elements (e.g., statements). Le and Lo's approach is the first study that predicts the effectiveness of automated debugging tools. To a certain extent, our proposed solution in predicting effectiveness of IR-based bug localization instances is a downgrade of Le and Lo’s approach from spectrum-based bug localization to IR-based bug localization. On the other hand, Le and Lo's approach uses features extracted from program execution traces and suspiciousness scores. In IR-based bug localization setting, there is no execution traces; thus we can only run Le and Lo's approach on features that are extracted from suspiciousness scores of files. We use this approach as a baseline to compare with APRILE+. We denote this baseline as SVMExt,Score. Furthermore, we also compare APRILE+ against APRILE.”

Please refer to comment 1.5 for additional details added to the description of RQ2

1. **Comment**: “For RQ5, how is this different from RQ1?”

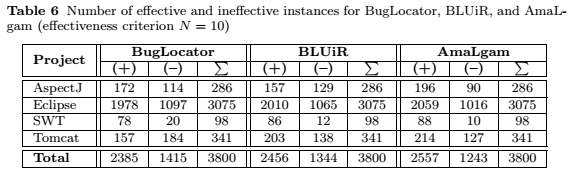
**Response**: Thank you for the question. We have merged RQ5 into RQ1.

We updated the description of RQ1 in Section 4.2 as follows:

“RQ1: How good is the performance of APRILE+ when predicting the effectiveness of an IR-based bug localization tool?

Answer to this research question will shed light on the utility of APRILE+. To answer this research question, we use APRILE+ to predict the effectiveness of BugLocator (Zhou et al, 2012), BLUiR (Saha et al, 2013), and AmaLgam (Wang and Lo, 2014) to locate buggy files. For each bug report in our dataset, we set the effectiveness criterion N = 10 and predict if each ranked list produced by BugLocator, BLUiR, and AmaLgam is effective or not.”

We updated Section 4.3.1 with new results of APRILE+ when predicting effectiveness of BugLocator, BLUiR, and AmaLgam instances as follows:



“Table 6 shows the statistics of effective and ineffective bug localization instances output by BugLocator, BLUiR, and AmaLgam to localize faults for 3,800 bug reports in our dataset. With effectiveness criterion N = 10, there are more effective instances than ineffective instances in most of data and bug localization tools. Considering all three bug localization tools (i.e., BugLocator, BLUiR, and AmaLgam), effective instances of BugLocator, BLUiR, and AmaLgam are 2,385, 2,456, and 2,557 respectively, which account for 62.76%, 63.63%, and 67.29% of the total instances, respectively.

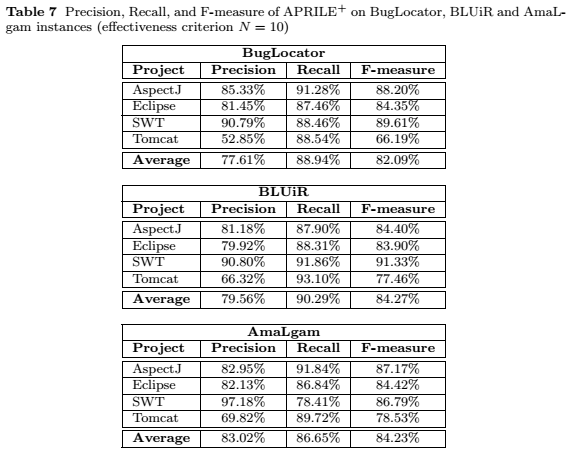


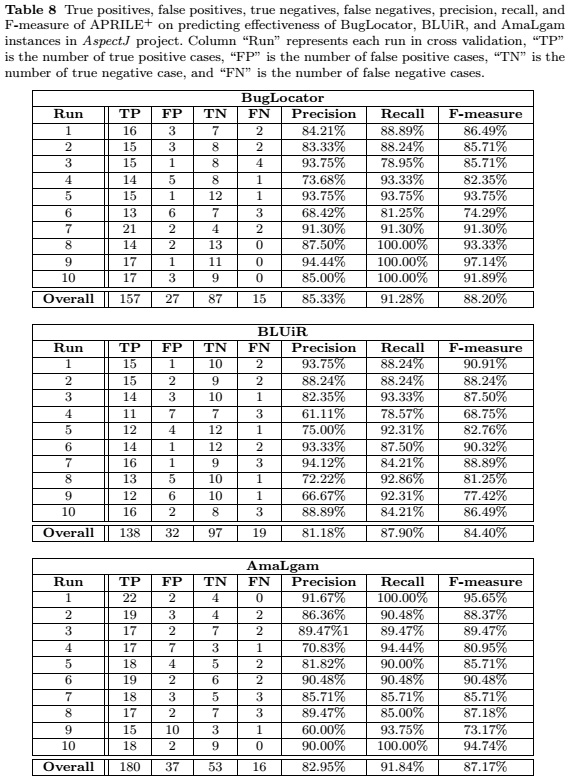
Table 7 shows the precision, recall, and F-measure of APRILE+ when predicting effectiveness of BugLocator, BLUiR, and AmaLgam instances in AspectJ, Eclipse, SWT, and Tomcat dataset. According to the table, APRILE+ achieves an F-measure of 66% or higher in predicting effectiveness of instances in AspectJ, Eclipse, SWT, and Tomcat project. Noticeably, APRILE+’s F-measure is up to 91.33% when predicting effectiveness of BLUiR instances in SWT. For Tomcat, APRILE+ achieves an F-measure of 66.19%, 77.46%, and 78.53%, respectively in predicting effectiveness of BugLocator, BLUiR, and AmaLgam instances. We find F-measures of APRILE+ in Tomcat dataset are smaller compared to AspectJ, Eclipse, and SWT dataset. This is because the proportion of effective instances to ineffective instances in Tomcat is less than the ones in AspectJ, Eclipse, and SWT. For the three bug localization tools (i.e., BugLocator, BLUiR, and AmaLgam), we note that APRILE+ achieves comparable average F-measures of 82.09%, 84.27%, and 84.23%, respectively. Furthermore, the F-measure results are comparable to or better than those achieved by other software analytics studies (Le and Lo, 2013; Seo and Kim, 2012; Shihab et al, 2013; Valdivia Garcia and Shihab, 2014).

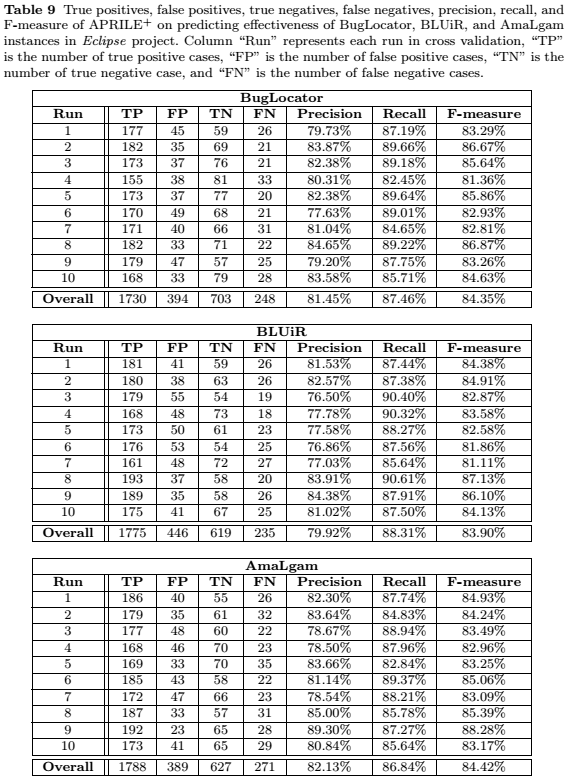
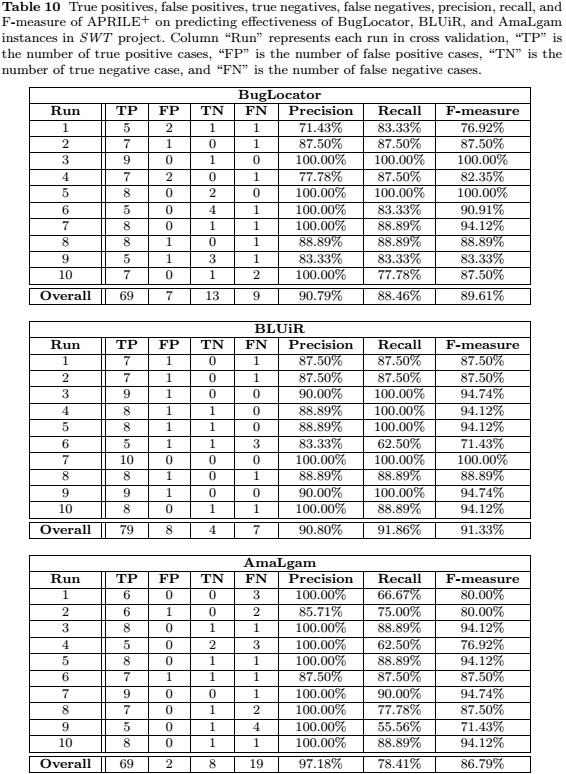
Table 8 shows detailed results of APRILE+ when predicting effectiveness of BugLocator, BLUiR, and AmaLgam instances in AspectJ dataset. APRILE+ achieves an F-measure of 88.20%, 84.40% and 87.17% for BugLocator, BLUiR, and AmaLgam, respectively. For BugLocator, APRILE+ is able to correctly predict 157 out of 172 effective instances, and 87 out of 114 ineffective instances. For BLUiR, our approach can correctly identify 138 out of 157 effective instances, and 97 out of 129 ineffective instances. For AmaLgam, it can correctly identify 180 out of 196 effective instances, and 53 out of 90 ineffective instances. Noticeably, without our tool, developers using BugLocator would inspect all instances (i.e., 286 instances), and they would be useless 39.86% of the time (i.e., 114 out of 286 instances are ineffective). By using our tool’s predictions, they are only useless 14.67% of the time (i.e., 27 out of 184 instances predicted as effective are actually ineffective). Similarly, developers using BLUiR would find that the instances are useless 45.1% of the time (i.e., 129 out of 286 instances are ineffective). By using our tool’s predictions, they are only useless 18.82% of the time (i.e., 32 out of 170 instances predicted as effective are actually ineffective). Last but not least, developers using AmaLgam would find that the instances are useless 31.47% of the time (i.e., 90 out of 286 instances are ineffective). By using our tool’s predictions, they are only useless 17.05% of the time (i.e., 37 out of 217 instances predicted as effective are actually ineffective). Furthermore, averaging across the three tools, the precision, recall, and F-measure of our approach for AspectJ are 83.15%, 90.34%, and 86.59%, respectively.

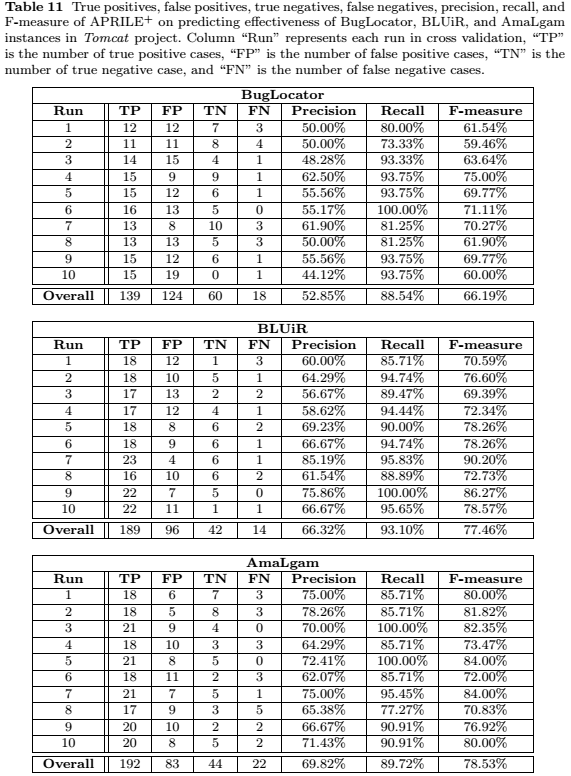
Table 9 shows detailed results of APRILE+ when predicting effectiveness of BugLocator, BLUiR, and AmaLgam instances in Eclipse. APRILE+ achieves an F-measure of 84.35%, 83.90% and 84.42% for BugLocator, BLUiR, and AmaLgam, respectively. For BugLocator, APRILE+ is able to correctly predict 1730 out of 1978 effective instances, and 703 out of 1097 ineffective instances. For BLUiR, our approach can correctly identify 1775 out of 2010 effective instances, and 619 out of 1065 ineffective instances. For AmaLgam, it can correctly identify 1788 out of 2059 effective instances, and 627 out of 1016 ineffective instances. Noticeably, without our tool, developers using BugLocator would inspect all instances (i.e., 3075 instances), and they would be useless of 35.67% of the time (i.e., 1097 out of 3075 instances are ineffective). By using our tool’s predictions, they are only useless 18.55% of the time (i.e., 394 out of 2124 instances predicted as effective are actually ineffective). Similarly, developers using BLUiR would find that the instances are useless 34.63% of the time (i.e., 1065 out of 3075 instances are ineffective). By using our tool’s predictions, they are only useless 20.08% of the time (i.e., 446 out of 2221 instances predicted as effective are actually ineffective). Last but not least, developers using AmaLgam would find that the instances are useless 33.04% of the time (i.e., 1016 out of 3075 instances are ineffective). By using our tool’s predictions, they are only useless 17.87% of the time (i.e., 389 out of 2177 instances predicted as effective are actually ineffective). Furthermore, averaging across the three tools, the precision, recall, and F-measure of our approach for Eclipse are 81.17%, 87.54%, and 84.22%, respectively.

Table 10 shows detailed results of APRILE+ when predicting effectiveness of BugLocator, BLUiR, and AmaLgam instances in SWT. APRILE+ achieves an F-measure of 89.61%, 91.33% and 86.79% for BugLocator, BLUiR, and AmaLgam, respectively. For BugLocator, APRILE+ is able to correctly predict 69 out of 78 effective instances, and 13 out of 20 ineffective instances. For BLUiR, our approach can correctly identify 79 out of 86 effective instances, and 4 out of 12 ineffective instances. For AmaLgam, it can correctly identify 69 out of 88 effective instances, and 8 out of 10 ineffective instances. Noticeably, without our tool, developers using BugLocator would inspect all instances (i.e., 98 instances), and they would be useless of 20.41% of the time (i.e., 20 out of 98 instances are ineffective). By using our tool’s predictions, they are only useless 8.14% of the time (i.e., 7 out of 86 instances predicted as effective are actually ineffective). Similarly, developers using BLUiR would find that the instances are useless 12.24% of the time (i.e., 12 out of 98 instances are ineffective). By using our tool’s predictions, they are only useless 9.2% of the time (i.e., 8 out of 87 instances predicted as effective are actually ineffective). Last but not least, developers using AmaLgam would find that the instances are useless 10.20% of the time (i.e., 10 out of 98 instances are ineffective). By using our tool’s predictions, they are only useless 2.82% of the time (i.e., 2 out of 71 instances predicted as effective are actually ineffective). Furthermore, averaging across the three tools, the precision, recall, and F-measure of our approach for SWT are 63%, 90.45%, and 74.06%, respectively.

Table 11 shows detailed results of APRILE+ when predicting effectiveness of BugLocator, BLUiR, and AmaLgam instances in Tomcat. APRILE+ achieves an F-measure of 66.19%, 77.46%, and 78.53% for BugLocator, BLUiR, and AmaLgam, respectively. For BugLocator, APRILE+ is able to correctly predict 139 out of 157 effective instances, and 60 out of 184 ineffective instances. For BLUiR, our approach can correctly identify 189 out of 203 effective instances, and 42 out of 138 ineffective instances. For AmaLgam, it can correctly identify 192 out of 214 effective instances, and 44 out of 127 ineffective instances. Noticeably, without our tool, developers using BugLocator would inspect all instances (i.e., 341 instances), and they would be useless of 53.96% of the time (i.e., 184 out of 341 instances are ineffective). By using our tool’s predictions, they are only useless 47.14% of the time (i.e., 124 out of 263 instances predicted as effective are actually ineffective). Similarly, developers using BLUiR would find that the instances are useless 40.47% of the time (i.e., 138 out of 341 instances are ineffective). By using our tool’s predictions, they are only useless 33.68% of the time (i.e., 96 out of 285 instances predicted as effective are actually ineffective). Last but not least, developers using AmaLgam would find that the instances are useless 37.24% of the time (i.e., 127 out of 341 instances are ineffective). By using our tool’s predictions, they are only useless 30.18% of the time (i.e., 83 out of 275 instances predicted as effective are actually ineffective). Furthermore, averaging across the three tools, the precision, recall, and F-measure of our approach for Tomcat are 63%, 90.45%, and 74.06%, respectively.”







1. **Comment** : “5.3.1 For RQ1 can you translate what this means for the user? So, before, the user would look at the results and they'd be useless X% of the time. Now, when using our algorithm, they are only useless Y% of the time :)”

**Response**: Thank you for the advice.

Note that in this revision, we merge RQ5 into RQ1, and change default effectiveness criterion to N=10 (please refer to comment 2.22, 2.24, 2.27, and 2.28 for further details). Thus, our explanation follows the updated results for RQ1.

Without our tool, developers would inspect all BugLocator instances (i.e., 3,800 instances), and they would be useless 37.24% of the time (i.e., 1,415 out of 3,800 instances are ineffective). By using our tool’s predictions, they are only useless 20.85% of the time (i.e., 552 out of 2,647 instances predicted as effective are actually ineffective). In other word, X=37.24% and Y=20.85%.

Without our tool, developers would inspect all BLUiR instances (i.e., 3,800 instances), and they would be useless 35.37% of the time (i.e., 1,344 out of 3,800 instances are ineffective). By using our tool’s predictions, they are only useless 22.68% of the time (i.e., 642 out of 2,831 instances predicted as effective are actually ineffective). In other word, X=35.37% and Y=22.68%.

Without our tool, developers would inspect all AmaLgam instances (i.e., 3,800 instances), and they would be useless 32.71% of the time (i.e., 1,243 out of 3,800 instances are ineffective). By using our tool’s predictions, they are only useless 18.65 % of the time (i.e., 511 out of 2,740 instances predicted as effective are actually ineffective). In other word, X=32.71% and Y=18.65 %.

We have updated Section 4.3.1 of this revision as follows:

*“*Table 8 shows detailed results of APRILE+ when predicting effectiveness of BugLocator, BLUiR, and AmaLgam instances in AspectJ dataset *… Noticeably, without our tool, developers using BugLocator would inspect all instances (i.e., 286 instances), and they would be useless 39.86% of the time (i.e., 114 out of 286 instances are ineffective). By using our tool’s predictions, they are only useless 14.67% of the time (i.e., 27 out of 184 instances predicted as effective are actually ineffective). Similarly, developers using BLUiR would find that the instances are useless 45.1% of the time (i.e., 129 out of 286 instances are ineffective). By using our tool’s predictions, they are only useless 18.82% of the time (i.e., 32 out of 170 instances predicted as effective are actually ineffective). Last but not least, developers using AmaLgam would find that the instances are useless 31.47% of the time (i.e., 90 out of 286 instances are ineffective). By using our tool’s predictions, they are only useless 17.05% of the time (i.e., 37 out of 217 instances predicted as effective are actually ineffective* …

*“*Table 9 shows detailed results of APRILE+ when predicting effectiveness of BugLocator, BLUiR, and AmaLgam instances in Eclipse ... *Noticeably, without our tool, developers using BugLocator would inspect all instances (i.e., 3075 instances), and they would be useless of 35.67% of the time (i.e., 1097 out of 3075 instances are ineffective). By using our tool’s predictions, they are only useless 18.55% of the time (i.e., 394 out of 2124 instances predicted as effective are actually ineffective). Similarly, developers using BLUiR would find that the instances are useless 34.63% of the time (i.e., 1065 out of 3075 instances are ineffective). By using our tool’s predictions, they are only useless 20.08% of the time (i.e., 446 out of 2221 instances predicted as effective are actually ineffective). Last but not least, developers using AmaLgam would find that the instances are useless 33.04% of the time (i.e., 1016 out of 3075 instances are ineffective). By using our tool’s predictions, they are only useless 17.87% of the time (i.e., 389 out of 2177 instances predicted as effective are actually ineffective) ...”*

“Table 10 shows detailed results of APRILE+ when predicting effectiveness of BugLocator, BLUiR, and AmaLgam instances in SWT … *Noticeably, without our tool, developers using BugLocator would inspect all instances (i.e., 98 instances), and they would be useless of 20.41% of the time (i.e., 20 out of 98 instances are ineffective). By using our tool’s predictions, they are only useless 8.14% of the time (i.e., 7 out of 86 instances predicted as effective are actually ineffective). Similarly, developers using BLUiR would find that the instances are useless 12.24% of the time (i.e., 12 out of 98 instances are ineffective). By using our tool’s predictions, they are only useless 9.2% of the time (i.e., 8 out of 87 instances predicted as effective are actually ineffective). Last but not least, developers using AmaLgam would find that the instances are useless 10.20% of the time (i.e., 10 out of 98 instances are ineffective). By using our tool’s predictions, they are only useless 2.82% of the time (i.e., 2 out of 71 instances predicted as effective are actually ineffective)* …”

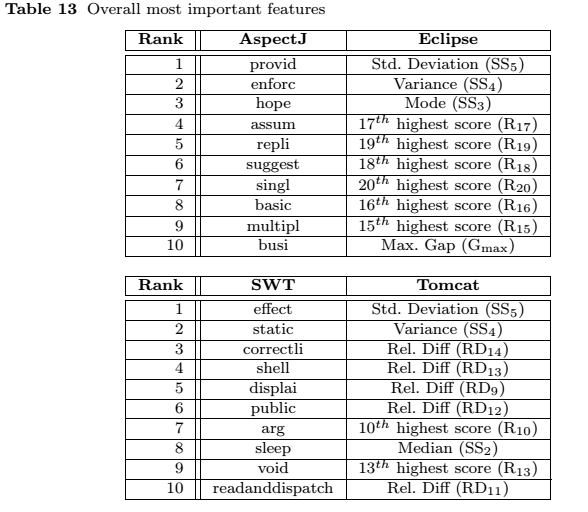
“Table 11 shows detailed results of APRILE+ when predicting effectiveness of BugLocator, BLUiR, and AmaLgam instances in Tomcat … *Noticeably, without our tool, developers using BugLocator would inspect all instances (i.e., 341 instances), and they would be useless of 53.96% of the time (i.e., 184 out of 341 instances are ineffective). By using our tool’s predictions, they are only useless 47.14% of the time (i.e., 124 out of 263 instances predicted as effective are actually ineffective). Similarly, developers using BLUiR would find that the instances are useless 40.47% of the time (i.e., 138 out of 341 instances are ineffective). By using our tool’s predictions, they are only useless 33.68% of the time (i.e., 96 out of 285 instances predicted as effective are actually ineffective). Last but not least, developers using AmaLgam would find that the instances are useless 37.24% of the time (i.e., 127 out of 341 instances are ineffective). By using our tool’s predictions, they are only useless 30.18% of the time (i.e., 83 out of 275 instances predicted as effective are actually ineffective)* ...”

1. **Comment**: “5.3.3 I don't want to see the top 5 for each category. I want to see the top 10 overall. I don't care what category they came from as much as I do which ones are best. My guess is that the metadata features are not in the top 10 overall.”

**Response**: Thank you for the advice. We have computed the top 10 features overall. We updated the description of research question 3 in Section 4.2 as follows:

“…If a feature has a Fisher score of zero, then that feature does not help to discriminate effective IR-based bug localization instances from ineffective ones. On the other hand, a feature is very discriminative if its Fisher score is much greater than zero. In this research question, we investigate the overall most discriminative features for each software project (i.e., AspectJ, Eclipse, SWT, and Tomcat).”

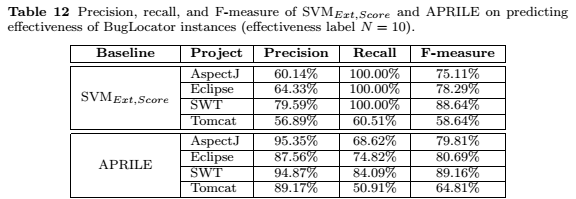
Note that in this revision we change the default effectiveness criterion to N=10 following advice from comment 2.27, and include new Tomcat dataset following comment 3.5. We updated the result of research question 3 in Section 4.3.3 as follows



“Table 13 shows the top-10 features with highest Fisher scores across the four categories of features. According to the table, we find that the score features and text features are the most discriminative ones. In particular, text features are more important for AspectJ and SWT, i.e., top-10 features with the highest Fisher scores for AspectJ and SWT are text features. We believe that there is a correlation between effectiveness labels of bug localization instances and occurrences of words in AspectJ and SWT’s textual bug reports, i.e., many effective and ineffective instances are likely described with different sets of words. On the other hand, score features are more discriminative in Eclipse and Tomcat as most of the features in the top-10 are from the suspiciousness score category. This is likely because the differences between the suspiciousness scores of faulty and non-faulty files in Eclipse and Tomcat are significant enough to distinguish effective from ineffective instances. Overall, text features are important for the AspectJ and SWT dataset, and score features are the most important for the Eclipse and Tomcat dataset.”

1. **Comment**: “5.3.4 I feel that you are being overly harsh on yourselves previously by using N = 1. I would use N = 10 in RQ1 and RQ2. If I knew that, with a 87.4 f-measure, I could find the answer in the top ten results, I'd be likely to use that tool. I would include that f-measure in the abstract, as it is close to a number that would make this approach usable in practice.”

**Response**: Thank you for the advice. We change the default effectiveness criterion to N=10. Please refer to comment 2.24, 2.26, and 3.5 for the changes in results of RQ1 and RQ3. We update the results of RQ2 in Section 4.3.2 as follows:



“Table 12 shows precision, recall, and F-measure of SVMExt,Score, and APRILE when predicting effectiveness of BugLocator instances with effectiveness criterion N = 10. Comparing the F-measures of SVMExt,Score and those of APRILE+ (shown in Table 7), we can note that APRILE+ outperforms SVMExt,Score on all datasets by up to 17.43%. Similarly, comparing the F-measures of APRILE against that of APRILE+, we can note that APRILE+ outperforms APRILE on all datasets by up to 10.51%.”

1. **Comment:** “5.3.5 I'd use N = 10.”

**Response**: Thank you for the advice. Please refer to our response in comment 2.24, 2.26, and 2.27.

1. **Comment**: “5.4 For threats to validity you mention implementation errors as the main problem. Do you have Unit Tests? Do you have Automated Tests? Do you have a redundant implementation? If not, I feel you must state these issues and/or open source your experimental code so it can be inspected.”

**Response**: Thank you for the advice. We have updated the threats to validity section (Section 4.4) as follows:

“… We have carefully rechecked our implementation several times, but there may still be errors that we do not notice. Admittedly, our system is a research prototype and not a mature industrial tool. Similar like other research prototypes, we do not have a comprehensive set of test cases to check its correctness…”

In the future, we are planning to release our implementation to the public, possibly integrated to an IDE and presented as a tool paper in a conference.

**Reviewer #3:**

1. **Comment**: “While, on the one hand, I found the paper clearly written and motivated, and with a good empirical evaluation of the approach, **I found this work a little bit marginal in terms of extension.** Essentially, the main novelty of the EMSE paper compared with the previous ISSRE 2014 paper consists of the use of ensemble classifiers to perform a bug localization effectiveness prediction. Noteworthy, your results clearly show that the ensemble classifiers work better than the simple classifiers being used by APRILE. This is fine, however using ensemble classifiers and bagging in particular is state-of-the-practice in machine learning, and is also **becoming quite popular** in software engineering applications too**. Can you please clarify what is the novelty point here**, i.e., (i) Is there anything novel and better than ensemble classifiers we could use from popular machine learning libraries or tools like Weka or R? (ii) Is there anything of the proposed approach that is peculiar of the software engineering problem being tackled here (i.e. predicting the effectiveness of IR-based bug localization tools) such that out-of-the-box ensemble classifiers cannot be used?”

**Response**: Thank you for the advice. The novelty points of our proposed approach (i.e., APRILE+) are as follows:

1. Although bagging is quite popular, it has not been used for the problem that we are addressing. We are the first to apply bagging to predict the effectiveness of IR-based bug localization instances.
2. Although the individual parts of APRILE+ are not novel, the composition is novel. Not all algorithms can be composed with APRILE and result in the improvement that we achieve. We have investigated many compositions and find a working one.
3. The features that we extract to characterize IR-based bug localization instances are peculiar of the software engineering problem being tacked (i.e., the prediction of the effectiveness of IR-based bug localization tools).

We have added the following text in the paper to highlight our contribution in view of the existing body of work in machine learning:

“Compared to the existing body of work in machine learning, our approaches (APRILE+ and APRILE) are novel in the following aspects:

1. Although bagging based methods are quite popular, they have not been used for predicting effectiveness of bug localization instances. We are the first to apply a bagging based method to predict effectiveness of IR-based bug localization instances.
2. Although individual parts of APRILE+ are not novel, the composition is novel. In fact, not all classification algorithms can be composed with APRILE, and result in improvement as we have achieved. We have investigated many compositions and find a working one.
3. The features that we extract to characterize IR-based bug localization instances are peculiar of the software engineering problem being tacked (i.e., the prediction of the effectiveness of IR-based bug localization tools)”
4. **Comment**: “One concern I also have about the previously published technique, i.e., APRILE, is about the use of multiple SVM classifiers on different sets of features, and their aggregation as for equation (1). Is there any specific reason for that? Intuitively, I would have just considered all features, performed (see below) a feature selection if needed, and then use a single classifier over all features. Probably you have reasons not to do so, however, it would be good to clearly justify that.”

**Response**: Thank you for the advice.

We divide the features into 4 categories as each category of features captures a specific characteristic of bug localization instances. Each of them is distinctive and different from the others. For example, score features capture numeric properties of the output suspiciousness scores, which are different from information captured by text features, topic model features, and metadata features. Therefore, each prediction model inferred from a category of features has its own prediction power leveraging on a specific characteristics of bug localization instances. Equation (1) combines these models together to formulate APRILE by tuning each model’s contribution (i.e., parameter) in order to maximize the accuracy. We do not mix all the features together and learn a single model as the many features may interfere with one another making it harder to learn a good discriminative model, or one group of features may dominate the rest in an unbalanced way, and in the end result in a poorer effectiveness.

We have updated Section 3.2.2 (APRILE’s Effectiveness Prediction Model) with our explanation:

“**Discussion**: We divide the features into 4 categories as each category of features captures a specific characteristic of bug localization instances. Each of them is distinctive and different from the others. For example, score features capture numeric properties of the output suspiciousness scores, which are different from information captured by text features, topic model features, and metadata features. Therefore, each prediction model inferred from a category of features has its own prediction power leveraging on specific characteristics of bug localization instances. Equation 2 combines these models together to formulate APRILE by tuning each model’s contribution (i.e., parameter) in order to maximize the accuracy. We do not mix all the features together and learn a single model as the many features may interfere with one another making it harder to learn a good discriminative model, or one group of features may dominate the rest in an unbalanced way, and in the end result in a poorer effectiveness.”

1. **Comment**: “A further concern is about feature selection. Some machine learners per se do already feature selection, some others (e.g. SVM) do not. In this paper, you indeed assess the contribution of different features to the prediction in RQ3. On the one hand, this could be interesting to identify features that could be used by IR-based bug localization tools. However, I'm really wondering why this analysis wasn't also done beforehand, and possibly using machine learning feature selection techniques, to build models that only include features that better help to perform the prediction.”

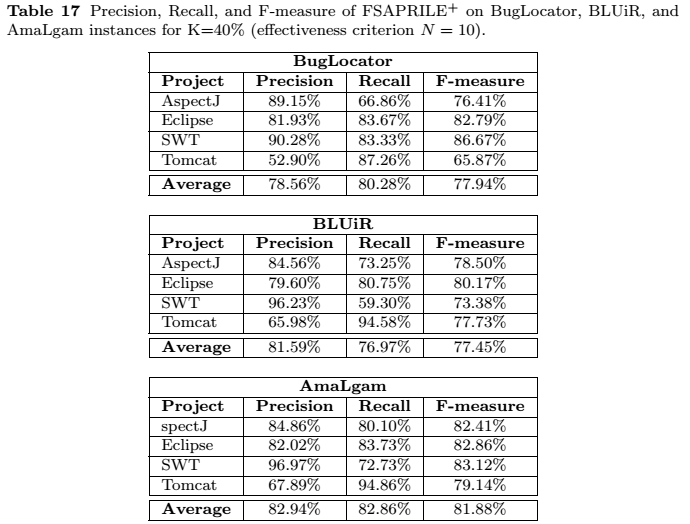
**Response**: Thank you for the advice. We extend our empirical evaluation by using feature selection in our proposed approach. In particular, we select top-N features with highest Fisher scores in each category to construct APRILE+’s component prediction models. Then, we inspect the precision, recall, and F-measure of APRILE+ when feature selection is applied.

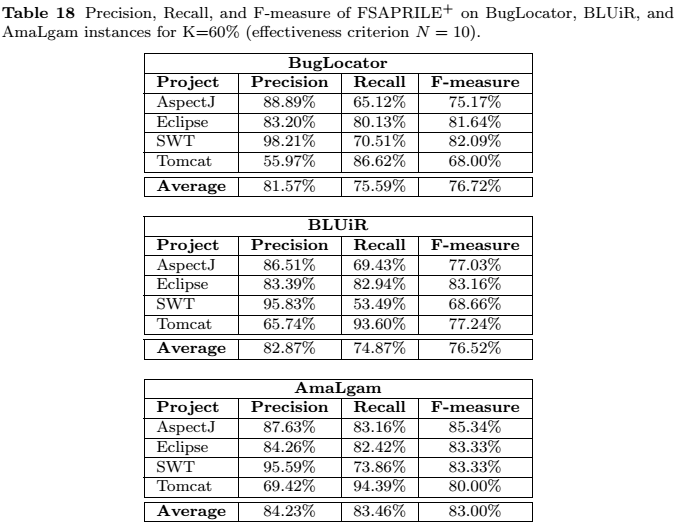
We create research question 6 in Section 4.2 of the current revision as follows

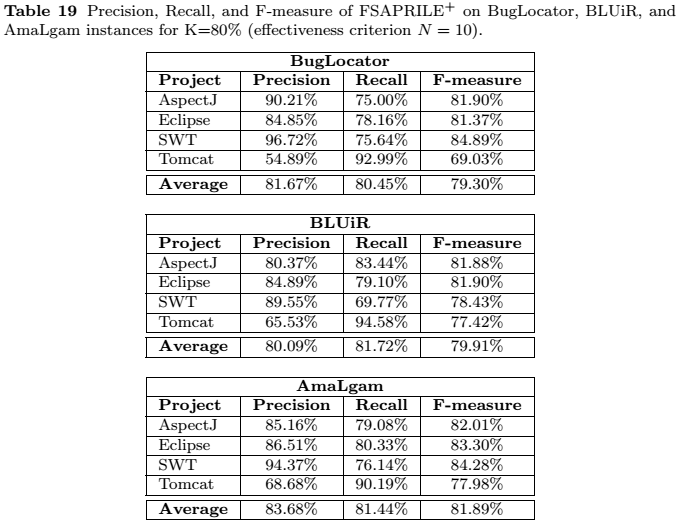
“RQ6: How good is the performance of APRILE+ if only the most discriminative features are selected to construct prediction models?

Feature selection techniques are proposed to improve the accuracy of classification models. However, by default, feature selection is not integrated into APRILE+. Thus, in this research question, we deploy features selection to our proposed approach by selecting the top K percent features with highest Fisher scores in each category to construct APRILE+’s prediction models. We refer to APRILE+ deployed with feature selection as FSAPRILE+. We use FSAPRILE+ to predict the effectiveness of BugLocator’s instances with K \in {40%, 60%, 80%}, and compare its precision, recall, and F-measure to those of APRILE+.”

We use the following text to describe the results in Section 4.3.6 of the current revision:







“4.3.6 RQ6: APRILE+ with Feature Selection

Tables 17-21 show the average precision, recall, and F-measure of FSAPRILE+ in various settings. According to the table, FSAPRILE+ achieves average F-measures of 77% or higher. Compared to the effectiveness of APRILE+ shown in Table 7, the F-measure of FSAPRILE+ is lower (for k=.. to …), but remains comparable to or better than those achieved by other software analytics studies (Le and Lo, 2013; Seo and Kim, 2012; Shihab et al, 2013; Valdivia Garcia and Shihab, 2014). For k=… and …, the F-measure of FSAPRILE+ is comparable to that of APRILE+. <the new one>”

1. **Comment**: “Actually, going on the results concerning the analysis of features themselves, I found such results pretty inconclusive (but ok, maybe this is itself a result), i.e. except for some features the set of relevant features really vary from project to project. This perhaps because the number of investigated projects is small, or perhaps because the characteristics of the issue reports for such projects are quite different. A further discussion on this point would be useful, because results seem to tell that very likely such a kind of model won't be able to work cross-project.”

**Response**: Thank you for the advice. In the revision, we updated Section 4.3.3 of the current revision (i.e., Section 5.3.3 in the previous revision) with further discussion on important features for each project as follows:

“**Discussion**: Table 13 indicates the most important features for each project is considerably different from one another. Noticeably, the overall most discriminative features for AspectJ and SWT are all text features, which are different from important features for Eclipse and AspectJ (see Table 13). The difference is due to the diverse characteristics of issue reports and source code files between Eclipse and Tomcat against the other two projects (i.e., AspectJ and SWT).”

We have also included Research Question 6 that inspects APRILE+ in cross-project setting. Please refer to our response to comment 1.7 for further details.

1. **Comment**: “Therefore, one possibility to further expand the paper and make it substantially different from the previous one could be:

- Expanding the dataset being used.

- Above all: experiment to what extent could the proposed technique be applied cross-projects.”

**Response**: Thank you for the advice. We extended the experiments by expanding the dataset to include 341 more bugs from Tomcat. We also evaluate the effectiveness of APRILE+ on cross-project setting by adding a new research question (i.e. research question 5 in this revision).

Please refer to our responses to comment 2.21 for details of the new Tomcat dataset. We have updated the results which answer all the research questions (in Sections 4.3.1, 4.3.2, 4.3.3, 4.3.4, 4.3.5, and 4.3.6 of this revision) to incorporate the new Tomcat dataset. Since the total number of bugs in the dataset, the average precision, recall, and F-measure are changed, we have updated the various relevant sections accordingly. For example, a relevant paragraph in the conclusion section has been updated as follows:

“… We refer to the extension of APRILE as APRILE+. We evaluate APRILE+ to predict the effectiveness of state-of-the-art bug localization techniques applied on a dataset of 3,800 bugs from AspectJ, Eclipse, and SWT. Our approach can achieve an average precision, recall, and F-measure of 77.61%, 88.94%, and 82.09%, respectively. Furthermore, APRILE+ outperforms a baseline based on the approach proposed by Le and Lo (Le and Lo, 2013; Le et al, 2014a) and APRILE by up to a 17.43% and 10.51% increase in F-measure, respectively.”

For the results of our newly added experiment, which evaluate to what extent the proposed technique could be applied cross-project setting, please refer to our response to comment 1.7.

1. **Comment**: “Other comments:

- The first sentence in the intro "In software development bugs are prevalent" is generic and not supported by any reference. I would just drop or rephrase it as it does not add anything

- Page 2 line 44: lose -> loose”

**Response**: Thank you for the advice. We have removed the sentence *“In software development, bugs are prevalent”*, and updated the first few sentences of the Introduction section as follows:

“Software debugging is an important task to maintain software quality. However, debugging is an expensive task …”

We have also changed “lose” to “loose” in page 2.

1. **Comment**: “- I found the background on IR-based bug localization (section 2.1) useful, whereas I did not find the very short descriptions of SVM, RBF, and Topic modeling very informative. Since each technique is described in few lines of text, I would rather shortly introduce each of them where it is used (e.g. LDA when the topic models features are described, SVM and RBF when APRILE and APRILE+ are introduced)”

**Response**: Thank you for the advice. We removed the short descriptions of SVM (Section 2.2) RBF (Section 2.3), and topic modeling (Section 2.4) in this revision. As recommended, we shortly introduce SVM and RBF when APRILE and APRILE+ are introduced, and LDA when topic features are introduced as follows.

SVM in Section 3.2.2:

“… APRILE's final prediction model contains four internal components which analyze score, text, topic model, and metadata features. Each component consists of a prediction model that specializes in a particular feature category. For example, score component only analyzes the score features, text component only analyzes the text features, and so on. *We use Support Vector Machine (SVM) to train the prediction model of each component. SVM is a popular classification algorithm that has been shown effective for many kinds of problems (Han and Kamber, 2006). It represents data instances as points in a multi-dimensional space where each feature is a dimension. It then separates data instances from different classes by finding a multi-dimensional hyperplane that best separates them. This hyperplane is often called the maximum marginal hyperplane (MMH). The underlying function (i.e., kernel) that defines the plane itself can be customized. The commonly used one is the linear kernel. In this work, we use SVM with linear kernel to build prediction models and consider a bug localization instance as a point in a multi-dimensional space. SVM is used to find the plane that separates effective bug localization instances from ineffective ones.*

In the training phase, SVM algorithm takes as input features of training bug localization instances whose effectiveness are known and learns a prediction model …”

RBF in Section 3.3.1:

*“*A radial basis function (RBF) network is a type of artificial neural network that can be used for supervised learning problems including classification (Broomhead and Lowe, 1988; Mitchell, 1997). The goal of a RBF network is to learn to convert a set of inputs into an output given a set of labeled examples. A RBF network consists of several layers, namely the input layer, hidden layer, and output layer, where each layer consists of a set of nodes. The nodes (or neurons) in the input layer corresponds to the set of inputs. The nodes in the hidden layer implements a set of radial basis functions (i.e., Gaussian functions) which convert a set of inputs into intermediary outputs. The node in the output layer sums up the intermediary outputs generated by the hidden layer. In the training process, the weights of edges connecting nodes in the input layer to nodes in the hidden layer are first determined. Next, the weights of edges connecting nodes in the hidden layer to nodes in the output layer are determined. RBF network can be trained in a short amount of time and it has good performance for various classification problems.”

Topic modeling in Section 3.1.3:

“Topic modeling is a technique to discover latent topics in a collection of documents. These latent topics are inferred based on the occurrences of words in the documents. One of the most popular topic modeling techniques is Latent Dirichlet Allocation (LDA) (Blei et al, 2003). LDA posits that each document is a mixture of topics and each word in the document is associated to a topic. Given a document, LDA generates its topic distribution, which corresponds to the probability of each topic to be assigned to the document. We apply Latent Dirichlet Allocation (LDA) (Blei et al, 2003) to extract a number of features from bug reports …”

1. Table 1: 18 feature -> 18 features

**Response**: Thank you the advice. We have changed “18 feature” in Table 1 to “18 features”.

1. **Comment**: “- Section 3.2.3: As you surely know LDA is quite sensible to the calibration of its parameters. First, it would be good to provide a justification of why you have chosen 5, 10, and 15 as possible numbers of topics (k). Also, it is not true that LDA only requires as input parameter the k. It also requires the smoothing coefficients for the topic distributions and word distribution (into topics), i.e. the alpha and beta coefficient, and, last, but not least, for some LDA implementations the number of Gibbs' iterations. Please at least report the used values of such parameters, and discuss in the threats to validity possible issues concerning a sub-optimal choice of such parameters.”

**Response**: Thank you for the advice.

We have added a clear description of all input parameters of LDA and the values that we use for these parameters as follows:

Section 3.1.3:

“… We apply Latent Dirichlet Allocation (LDA) (Blei et al, 2003) to extract a number of features from bug reports. LDA accepts a number of input parameters:

1. k, which is the number of topics that should be inferred from the input documents (i.e., bug reports).

2. α, which affects the topic distributions per documents. Higher values of α make topics more uniformly distributed (i.e., better smoothing of topics) in each document.

3. β, which governs the word’s distributions per topics. Higher values of β lead to words more uniformly distributed in every topic.

4. n, which is the number of Gibbs’s iterations i.e., the number of times Gibbs sampler is invoked (Blei et al, 2003).

…"

And in Section 4.4.1:

“**Latent Dirichlet Allocation**: We employ Stanford Topic Modeling Toolbox8 to train topic models. We set the number of topics to 5, 10, and 15. We use these numbers of topics to capture high-level concepts that are shared by many bug reports – the less the number of topics, the more abstract (or high-level) the topics are. For the other parameters of LDA we use the default settings of Stanford Topic Modeling Toolbox: α = 0.01, β = 0.01, and number of iterations = 1000.”

8http://nlp.stanford.edu/software/tmt/tmt-0.4/

We have also added several paragraphs in the threats to validity section on possible issues concerning the sub-optimal choices of such parameters:

“LDA is quite sensitive to the calibration of its parameters (i.e., alpha, beta, number of iterations, and number of topics). In this paper, we set the number of topics as 5, 10, and 15. We also fixed the alpha, beta, and number of iterations to the default values of Stanford Topic Modeling Toolbox v0.4.0 implementation of LDA (i.e., alpha = 0.01, beta = 0.01, and number of Gibbs’ iterations = 1000). These values may not be optimal, and the result of APRILE and APRILE+ may improve when these parameters are optimized.”

1. **Comment**: “- Section 3.3: as I mentioned above, I wasn't convinced about using multiple classifiers. Even if you use them, why you perform a stepwise choice of the coefficients? Instead, you could have just learned them or used a meta-heuristic to choose them.”

Response: Thank you for the advice.

We plan to do an investigation of choosing the coefficients by learning approaches (e.g., by using learning to rank, meta-heuristics, or genetic algorithm) in a future work. We have updated the Conclusion and Future Work section as follows

“… Moreover, we plan to employ other approaches (e.g., learning to rank, meta-heuristics, or genetic algorithm) to tune coefficients of models in Equations 2 and 3 …”

1. **Comment**: “- Section 5.3.1: (totally optional) I’d suggest to report confusion matrices along with Table 7, other than reporting the values inline in the text”

**Response**: Thank you for the advice. We have reported information of true positives, false positives, true negatives, and false negatives in Table 8, 9, 10 of this revision. Please refer to our response to Comment 1.6 for further details.

1. **Comment**: “- Section 5.3.3: as I said it is not clear how such discussion could help, whether this could help to build better bug-localization tool, or to explain APRILE+ results. In any cases, I see quite a difference in terms of most important features across projects.”

**Response**: Thank you for the advice. Following comment 3.4, we have updated Section 5.3.3 with further discussion. Please refer to our response to Comment 3.4 for more details.

1. **Comment**: “- Section 5.4: some threats to internal validity could be due to the choice of the machine learning techniques being used as well as to the calibration of LDA”

**Response**: Thank you for the advice. We have updated the Threats to Validity section as follows:

“… The other threat to external validity is our choice of classification algorithms to construct prediction models. We selected only two algorithms: Support Vector Machine and Radial Basis Function Network. There are still other potential algorithms that are more accurate than these two algorithm. In the future, we plan to integrate more classification algorithms in APRILE+. LDA is quite sensitive to the calibration of its parameters (i.e., α, β, number of iterations, and number of topics). In this paper, we set the number of topics as 5, 10, and 15. We also fixed the α, β, and number of iterations to the default values of Stanford Topic Modeling Toolbox v0.4.0 implementation of LDA (i.e., α = 0.01, β = 0.01, and number of Gibbs iterations = 1000). These values may not be optimal, and the result of APRILE and APRILE+ may improve when these parameters are optimized…”