Jonas Paul Winkler, Jannis Grönberg, Andreas Vogelsang

Optimizing for Recall in Automatic Requirements Classification: An Empirical Study

Accepted for 27th IEEE International Requirements Engineering Conference, http://re19.ajou.ac.kr/.

Winkler, Jonas Paul; Grönberg, Jannis; Vogelsang, Andreas (2019): Optimizing for Recall in Automatic Requirements Classification: An Empirical Study. 27th IEEE International Requirements Engineering Conference (RE'19).

Optimizing for Recall in Automatic Requirements Classification: An Empirical Study

Jonas Paul Winkler
Technische Universität Berlin
Berlin, Germany
jonas.winkler@tu-berlin.de

Jannis Grönberg
Technische Universität Berlin
Berlin, Germany
jannis.r.groenberg@campus.tu-berlin.de

Andreas Vogelsang
Technische Universität Berlin
Berlin, Germany
andreas.vogelsang@tu-berlin.de

***Abstract*—Using Machine Learning to solve requirements engineering problems can be a tricky task. Even though certain algorithms have exceptional performance, their recall is usually below 100%. One key aspect in the implementation of machine learning tools is the balance between recall and precision.**

Tools that do not find all correct answers may be considered useless. However, some tasks are very complicated and even requirements engineers struggle to solve them perfectly. If a tool achieves performance comparable to a trained engineer while reducing her workload considerably, it is considered to be useful.

One such task is the classification of specification content elements into requirements and non-requirements. In this paper, we analyze this specific requirements classification problem and assess the importance of recall by performing an empirical study. We compared two groups of students who performed this task with and without tool support, respectively.

We use the results to compute an estimate of β for the F_β score, allowing us to choose the optimal balance between precision and recall. Furthermore, we use the results to assess the practical time savings realized by the approach.

By using the tool, users may not be able to find all defects in a document, however, they will be able to find close to all of them in a fraction of the time necessary. This demonstrates the practical usefulness of our approach and machine learning tools in general.

***Index Terms*—Empirical research, controlled experiment, machine learning, automation**

I. Introduction

Recent advances in natural language processing and machine learning led to an increasing number of approaches that try to solve requirements engineering tasks by some form of automation [1]. In almost all cases, the automatic approaches are not able to solve the tasks perfectly, i.e., with 100% precision and 100% recall. Therefore, most authors argue that the approaches aim to support the requirements engineer in performing a task. However, it is not clear what quality a tool must achieve to justify this claim. An anecdotal example is given by Berry [2] who argues that for problems where all correct answers have to be found, every tool with recall below 100% is useless because the requirements engineer needs to inspect the entire document anyway to identify the (few) correct answers that the tool missed. Even for cases not as extreme as this, recall is usually considered more important for automating requirements engineering tasks than precision [2]. However, many authors still use the F_1 score to optimize and evaluate their approaches, which weighs precision and recall equally.

One specific task in the requirements engineering process is the classification of specification content elements into requirements and non-requirements. While requirements are the basis for tests and define what is legally binding for the contractor, non-requirements may contain explanatory information, examples, as well as figures, tables and references to other documents. We have proposed an approach that automatically classifies natural language sentences in requirements specifications into requirements and non-requirements [3]. In our paper, we used F_1 to evaluate the performance of our approach. In this paper, we instead follow the suggestions of Berry [2] and derive a reasonable value for β for the problem of finding defects in requirements/non-requirements classifications. We used our tool to identify classification defects in already labeled requirements specifications and performed a controlled experiment with 16 students to compare the performance of two groups: one scanned the specifications manually for classification defects ("‘manual group"’), while the other was supported by a tool ("‘tool group"’). Based on this experiment, we were able to derive the following results:

- The tool group achieved a higher recall for finding defects (0.51 on average) than the manual group (0.39), even though the highest achievable recall for the tool group was limited by the capabilities of the tool (recall of 0.84 and 0.66 on the documents used for the experiment).
- We determined $\beta \approx 6.2$ by comparing the time for a human to manually find a true positive in the original documents and the time for a human to reject a tool-presented false positive [2]. The value indicates that recall is more important for the examined problem.
- Using F_1 to tune the tool results in a recall of 0.83, a precision of 0.81, and a summarization of 0.83. When using $F_{6.2}$ for tuning, the tool has a recall of 0.98, a precision of 0.42, and a summarization of 0.61. Based on our experiment, we know that these values represent a decent balance between recall and precision for the defect detection task.

The contributions of this paper can be summarized as follows:

- We picked up a claim about using F_β instead of F_1 for certain classification tasks in RE [2] and assessed this claim in an empirical study with real-world data, a real-

world problem, and students as a proxy for real-world engineers.
- Our main finding is that, in the given setting, finding defects with the help of a classification tool works better than working on the original data and that using F_β instead of F_1 for optimizing the tool makes sense and reduces the number of elements that need to be examined by a human by 61% (i.e., summarization).

Our results show that even non-perfect tools can improve RE tasks that have been performed manually so far. Tuning these tools with respect to an empirically determined β resulted in considerable time savings: Using our optimized classifier reduces the manual work from inspecting 100% of the elements in the data set to inspecting a subset comprising only 39% of the original elements, while assuring that 98% of all classification defects are located in that subset.

II. Background

A. Requirements Specifications, Requirements and Non-Requirements

In many requirements engineering (RE) processes requirements specifications are used to document the properties that a system has to exhibit in order to be accepted. Other purposes of requirements specifications are the derivation of test specifications and defining liability between stakeholders (i.e., what must be achieved to fulfill the contract). A *requirements specification* is a document that contains *content elements*. Content elements are used to structure a requirements specification. A content element may contain text, bullet points, tables, images, etc. Each requirement in a requirements specification is stored in a separate content element.

In addition to legally binding requirements, requirements specifications contain additional information, which we refer to as *non-requirements*. This includes examples and explanations, as well as figures and references to other documents. Each non-requirement is also stored in a separate content element. Although non-requirements are not requirements which must be fulfilled by the supplier, they provide background knowledge, which is crucial for understanding requirements and their context.

Explicit differentiation between requirements and non-requirements increases the quality of a requirements specification. Since further development steps depend on correct definition and documentation of requirements, an accurate differentiation between requirements and non-requirements is vital. For example, when a test specification is derived from a requirements specifications, this differentiation defines for which content elements a test case has to be created. Also, this differentiation defines which content elements have to be implemented by a supplier. Therefore, each content element is annotated with a *label*, which defines the content element either as requirement or non-requirement. At one of our industry partners, these labels are created and verified manually, which is time-consuming and error-prone. Whenever the label of a content element does not match the actual type of the content element, we refer to this content element as a *defect*. Adding the labels at a later stage, as well as finding and fixing possible defects is expensive since every content element has to be read and understood again.

B. Automatic Requirements Classification Tools

Classification tools represent a convenient solution to support a requirements engineer in classifying requirements. They can either be used to auto-classify unlabeled content elements or to review already labeled ones. In both cases, these tools do not operate alone, but rather reveal defects in content elements and suggest another label.

Automatic requirements classification tools are used to distinguish functional and non-functional requirements [4], identify bug reports and feature requests in app reviews [5], or group requirements according to topics [6]. There are several types of underlying classification approaches. Common approaches include decision trees, Naive Bayes classifiers or Support Vector Machines [7]. Furthermore, recent studies found that even simple convolutional neural networks (CNN) can achieve excellent results in multiple benchmarks for natural language classification tasks [8].

C. Automatic Classification of Requirements and Non-Requirements

We have developed a tool that is able to classify natural language content elements from requirements specifications into requirements and non-requirements [3]. The approach is based on convolutional neural networks and also offers a visualization component to help engineers understand the decisions of the classifier [9]. We also conducted an experiment to determine the usefulness of our tool [10]. During the experiment, participants were split up into two groups. Both groups had to edit two real-world requirements documents. One group was assisted by the tool, the other group performed the task without the tool. The accuracy of the tool was different for the two examined documents. While the tool detected defects in the first document with an accuracy of 82.6%, the accuracy in the second document was lower (75.8%). We stated the following main findings [10]:

- The accuracy of the tool has an impact on the defect correction rate. While the defect correction rate of the tool-assisted group was 11% higher in the document where the tool had a higher accuracy (48% with tool and 37% without tool), the defect correction rate was 21% lower for the tool-assisted group in the document where the tool had a lower accuracy (40% with tool and 61% without tool).
- Independent of the accuracy of the tool, the tool-assisted group introduced less new defects while reviewing the specifications.
- Participants missed more unwarned defects (i.e., false negatives) if they were assisted by a tool. 90% of defects without a warning from the tool were not corrected, while participants reviewing manually missed only 62%.

These results show that an optimal balance between precision and recall is crucial for a tool that aims to assist a requirements engineer. We reused our tool in the experiment to derive a reasonable value for β and afterwards tune the tool with respect to this value.

D. Calculating F_β to Evaluate and Optimize Classification Tools

The standard procedure to evaluate assistance tools is to use *precision* (1) and *recall* (2). *Precision* indicates the percentage of correct answers over all answers found by the tool:

$$P = \frac{TP}{TP + FP} \tag{1}$$

Recall indicates the percentage of correct answers found by the tool over all possible correct answers:

$$R = \frac{TP}{TP + FN} \tag{2}$$

The composition of both evaluation metrics is called *F-measure* (3):

$$F = 2 \times \frac{P \times R}{P + R} \tag{3}$$

We also use a fourth measure called *summarization*, which indicates by how much an original document is reduced [2]:

$$S = \frac{TN + FN}{TN + FN + TP + FP} \tag{4}$$

Berry [2] describes that in most use cases of tool assistance with natural language problems, the recall of a tool is significantly more important than precision. A tool with insufficient recall may be useless for the development of a highly complex system, since a human has to do the entire task manually anyway in order to find missing information. If a tool can not provide a recall close to 100%, a human working with the tool must at least achieve a recall better than a human without tool assistance.

Therefore, tool assistance in the requirements engineering domain needs to be evaluated by a weighted F-measure called the *F_β-measure* (5):

$$F_\beta = (1 + \beta^2) \times \frac{P \times R}{(\beta^2 \times P) + R} \tag{5}$$

Defining β as 1 results in *F_β-measure* as F_1. In this case the formula for F_β is equal to the formula for F. As β grows, the relevance of precision for computing F_β declines and F_β approaches the recall.

Choosing β determines the ratio by which recall is weighted higher than precision. According to Berry [2], β is calculated as follows. Given a document D and a tool t, β is the ratio of

- the average time that an average human needs to manually find a correct answer in D, and
- the average time that an average human needs to manually vet any potential answer that t returns.

An empirical study is necessary to determine these values for each use case and the results are bound to the tool t and generally not transferable to other tools or use cases. The denominator of β may also be calculated by estimating "‘the average time that an average human needs to manually determine whether or not any potential answer in D is a correct answer"’ [2]. This measure is not specific to any tool and may only be acquired during gold standard construction. However, as we want to optimize the tool, we chose to go with the tool-specific approach.

III. Research Design

As already discussed in Section II-D, β can only be calculated using empirical data. It is also task-specific and must be calculated individually for each task. We performed an adequate empirical study to determine β for the task of classifying specification content elements into requirements and non-requirements.

In this empirical study, two groups of students identified classification errors in two requirements specifications that we prepared specifically for this study. The first group performed the task manually, that is, without any support by a tool. The second group inspected only the elements containing defects as determined by the automatic classification tool. The second group also saw the suggested label for each defect as reported by the tool. By comparing the results of both groups against the gold standard, we were able to compare the performance of both groups and measure any improvements achieved by using the tool.

We organized our research according to these research questions:

- **RQ 1:** *Is there any performance difference between the two groups?* We assume that the group working with the output of the tool will perform better than the manual group. Since the tool group works with a smaller portion of the document (only the elements issued by the tool), they should be faster than the other group and still find a similar number of errors, or even more.
- **RQ2**: *What is the optimal β for the requirement/non-requirement classification task?* This value is calculated from data acquired during the experiment and can be used to determine the best ratio of precision and recall for the task.
- **RQ3**: *How big is summarization given β on typical requirements specification documents?* Summarization measures the ratio by which a requirements specification is reduced in size because a tool issues only the interesting elements (i.e., true positives and false positives). Higher summarization results in more saved time by requirements engineers.

In the following subsections, the experiment used to answer these questions will be described in detail. We followed the guidelines provided by Ko et al. [11] and Jedlitschka et al. [12].

TABLE I
EXPERIMENT DESIGN

	Group 1	Group 2
Session 1	Control Group (CG)	Treatment Group (TG)
Session 2	Treatment Group (TG)	Control Group (CG)

A. Tool Description and Preparation

The tool used for the experiment uses a natural language text classifier to decide the label of a content element. The classifier is built using a Convolutional Neural Network for Text Classification [8]. This network primarily consists of two layers. The first layer contains a set of *filters* which scan the input for patterns. The second layer associates these patterns with the labels. The network outputs probabilities for each label. During training, the network learns to recognize certain patterns and learns which patterns to associate with which label. Please refer to Kim [8] for further details.

The tool uses this network to find defects in a requirements specification. The tool has an adjustable *threshold* that controls how many detects are detected. If set to one, no defects are reported. If set to zero, every element is reported as a defect. A threshold close to one means that only defects with very high confidence are reported, wheres a threshold close to zero results in a defect set that includes all elements except those which are most likely correct.

Before we used the tool to run the experiment, we trained its internal classifier on a dataset containing 35000 pre-labeled content elements (20000 requirements and 15000 non-requirements). This dataset was constructed from real-world requirements specifications from the automotive domain, available at one of our industry partners. The documents used for the experiment were not included in the dataset. After training and performing 10-fold cross validation and adjusting the threshold to optimize F_1, the classifier achieved a recall of 0.83, a precision of 0.81 and a summarization of 0.83 on the dataset.

B. Experiment Design

We employed a two-by-two crossover design [13]. In this experiment design, two groups will perform a given task using two different methods. The treatment group will work with the output produced by the tool, whereas the control group will work on unfiltered documents and without tool support. In addition to that, the experiment consists of two sessions using two different documents. In both sessions, the treatment and control group will perform the same task. However, groups are switched between both sessions so that each participant produces data both with and without tool support. Table I outlines the design.

C. Participants

We conducted the experiment with students as part of a lecture series on requirements engineering in the automotive domain. The lecture was a second semester master course and the experiment was performed near the end of the semester. Therefore, the students had already acquired knowledge about topics such as requirements engineering in general, its application in the automotive context, involved processes, test engineering and requirements quality. However, their prior exposure to requirements engineering may differ and therefore some students may perform better at the given task than others. The design of the experiment helps to mitigate this issue, since every participant will contribute data to both the control and the treatment group. We did not collect any other demographic data about the students since our time was limited and additional data is not needed to answer our research questions.

We announced the experiment before and asked them to participate since a high number of participants is required for a better statistical evaluation of the experiment. We advertised the experiment as a chance to work with real-world requirements. However, only 16 students were present, which is about half of the number of students enrolled in the lecture.

D. Experiment Material

The requirements specifications used for the experiment were derived from actual work-in-progress specifications at our industry partner. We did not use original specifications due to several reasons:

- The specification of automotive systems are usually very large. Most system specifications consist of more than 1000 individual requirements plus additional content such as non-requirements and headings. It is not feasible to use such a long specification for empirical tests, since it would take the students multiple hours to complete the tasks, resulting in serious degradation of performance due to fatigue.
- Specifications usually consist of multiple abstraction levels. While requirements specifying the overall behavior of a system are quite easy to understand, requirements specifying very detailed hardware attributes (pinnings, signal specifications, bus interfaces) are not. Understanding these requires knowledge in the respective field, which the students probably do not have yet.
- The requirements specifications at our industry partner contain many sensitive information which should not be made public.

Therefore, we selected the specifications of two systems whose functionality is easy to understand. The *Wiper Control* (WWC) system incorporates the wipers, a control lever, a control unit and an optional rain sensor. The specification describes how these components work together. It is a system everyone should be familiar with. The *Hands-Free Access* (HFA) system is a novel system which allows the driver to open the trunk door by performing a kick motion towards the trunk door.

First of all, we reduced the size of the document by selecting sections from the document describing core functionality of the system. Both specifications have a section which describes the functions of the systems on a very high level. This choice excluded many technical content elements that were hard to understand and reduced the overall size of the dataset to a reasonable number.

Next, we reviewed the documents manually, identified defects and annotated each element with its correct label, i.e., we established a gold standard on both documents. Afterwards, we used the tool to generate predictions for all elements in both documents. The threshold of the tool was not adjusted and the tool achieved a recall of 0.84 on the Wiper Control document and a recall of 0.66 on the Hands-Free Access document.

Based on this data, we prepared two versions of each document: The first version was for the control group and contained all elements of the document, the original labels and no tool suggestions. The second version, for the treatment group, contained only the elements for which the tool proposed a different label. The elements in this version include the original label, as well as the label suggested by the tool.

Finally, we edited the text of the documents and replaced sensitive information such as corporate-internal names of systems, components, signals and values such as dimensions and voltages with dummy names and values. This was done last so the changes would not affect the automatic classification.

Examples of requirements and non-requirements from the final documents are provided below:

- [requirement] When the lever is moved from position 0 to position 1, the system shall start interval wiping.
- [requirement] The wiping functionality has to be paused during engine start.
- [non-requirement] The term front wiping speed refers to the rotation speed of the front wiping motor.
- [non-requirement] The contents of the signal SYSSIGNAL-2 are still subject to change.

Statistics about the final documents are provided in Table II. We assumed that it would take the students roughly 10 s to review a single element. This value was taken from previous study results [10]. Therefore, the participants should be able to complete the review in 20 minutes (Wiper Control) and 25 minutes (Hands-Free Access).

Tool summarization is very good with the default settings. 75% tool summarization indicates that the treatment group analyzed only one quarter of the total document. However, tool defect recall is particularly low on the Hands-Free Access specification and already indicates that the classifier needs tuning. Recall also effectively limits how many of all the defects the participants are able to find in the document.

E. Tasks

The tasks for this experiment were designed to mimic quality audits in practice. Therefore, we asked the participants to perform a full review of the document and fix any defects they find.

Participants of the control group were asked to read each individual element, determine its classification and correct the given classification if it does not match. Participants of the treatment group were asked to read each individual element as well and assess whether the tool-proposed correction is actually correct. If the participants thought the tool was correct, they marked the element using an “x”.

TABLE II
EXPERIMENT MATERIALS

	Wiper Control	Hands-Free Access
Total elements	115	147
Requirements	85	79
Non-requirements	30	68
Total defects	19	47
Defects per element	0.165	0.320
Defects in requirements	9	16
Defects in non-requirements	10	31
Tool returned warnings	25	37
Tool true positives	16	31
Tool summarization	25 / 115 = 78.3%	37 / 147 = 74.8%
Tool defect recall	16 / 19 = 0.842	31 / 47 = 0.660
Tool defect precision	16 / 25 = 0.640	31 / 37 = 0.838

F. Experiment Procedure

The experiment was divided into multiple sections outlined below. Since the experiment was conducted during the lecture, we had to make sure that its length would not exceed 90 minutes. This is also one of the reasons why we were unable to use longer documents for the experiment.

Introduction (25 min). We introduced the students to the problem of requirements and non-requirements classification. We provided examples of both classes and pointed out its importance in downstream engineering processes. We also introduced our research on assisted classification and especially highlighted how tools may help engineers save time or be more accurate. We introduced the experiment, our goals, and the tasks the students are supposed to do in the experiment.

Session 1 (20 min). During the first session, we assigned all students evenly to one of the two groups and handed out the Wiper Control specification. Each participant recorded his or her individual time by documenting start and end time. We did not allow communication between the students, since this would negatively impact the independence of the samples.

Session 2 (25 min). During the second session, we switched groups and repeated the process exactly as in the first session for the Hands-Free Access Specification.

Summary and outlook (15 min). After completing both sessions and collecting the results, we gave a quick overview of what we expect from the results and how these results will affect our research and the applicability of the machine-learning based tool in requirements engineering.

G. Evaluation Plan

The evaluation of the results is structured into three steps according to our three research questions.

Performance differences between the groups. Since the treatment group works with the direct output of the tool, we assume that their results will be better than the results of the control group and closer to the performance of the tool as presented in Table II. We will compare both groups using the following metrics.

The *Defect Detection Recall* measures how many defects a participant finds in the specification.

$$Recall_{DefectDetection} = \frac{DefectsCorrected}{TotalDefects}$$

In case of the treatment group, the Defect Detection Recall of a single participant cannot be higher than the recall of the tool as presented in Table II. The control group is theoretically able to detect all errors. Our hypothesis is that the recall of the treatment group is higher due to the focus on fewer elements and the tool suggestions.

The *Defect Detection Precision* measures how many changes of a participant actually fixed a defect.

$$Precision_{DefectDetection} = \frac{DefectsCorrected}{ElementsChanged}$$

We consider an element to be changed when the participant assigned a different label or when she accepted the suggestion by the tool. Changing a previously *correct* content element introduces a new defect to the specification. When precision is below 0.5, more new defects were introduced than defects fixed. We expect the precision of the treatment group to be higher due to the focus on fewer elements and the tool suggestions as well.

The *Time Per Element* measures how many seconds a participant spent on average to make a decision for one element in the specification. This includes reading the element, making the decision, and documenting the decision in the specification.

$$TimePerElement = \frac{EndTime - StartTime}{TotalElements}$$

Our hypothesis is that participants of the treatment group may need more time per element. In addition to reading the element, determining its classification and documenting the result, they also have to evaluate the tools suggestion against their own classification. However, total time per review should still be greatly reduced due to the significantly smaller number of elements in the review as measured by *Time Total*:

$$TimeTotal = EndTime - StartTime$$

Calculation of β. After performing a basic evaluation of the results as presented above, we calculate β as described in Section II-D:

$$\beta = \frac{TimeTotal(CG)}{DefectsCorrected(CG)} * \frac{1}{TimePerElement(TG)}$$

In this formula, the first part is the average time of a participant in the control group (CG) to identify and correct a defect. This also includes the time needed to read and dismiss correctly classified elements. The second part is the average time a participant in the treatment group (TG) needs to either accept or reject a single answer from the tool.

Calculation of F_β and summarization. The main advantage of our approach is that it may reduce the time of manual reviews. Therefore, we will estimate average summarization of the tool by using β and the following synthetic test:

The dataset that was used to train the classifier contains labels for all elements and the performance of the classifier is trained and tested using those labels. We assume that the labels in this dataset are correct. Therefore, the classifier should be able to identify elements on which the label was changed after training (i.e., identify elements which contain a defect). We introduce defects into the *test* set by changing the labels of randomly selected elements. The number of defects is defined by *Defects per element* as presented in Table II.

We evaluate the ability of the tool to detect these errors. The threshold of the tool is set so that F_β on the test set is highest. We will then measure summarization on the test set.

IV. Study Results

In this section, the results of our experiment will be presented. The results for recall, precision, time per element and total time may be found in Figures 1, 2, 3 and 4. Overall, 16 students participated in the experiment. A total of 32 reviews are available, 16 manual and 16 tool-assisted reviews. All of the students were able to complete both reviews in time. More details are available in Table III.

TABLE III
Experiment result overview

	CG	TG
Number of reviews	16	16
Elements inspected	2096	496
Elements changed	448	354
Defects corrected	216	274
Cumulative total time	18 609 s	6913 s

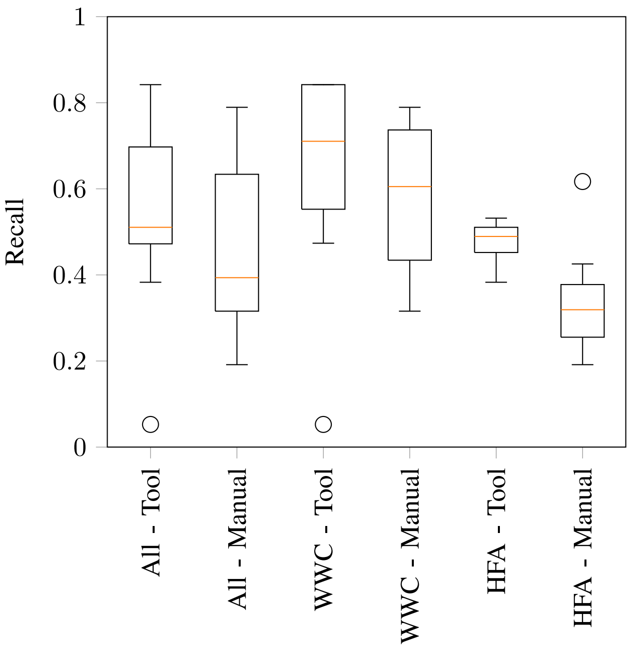

Fig. 1. Recall

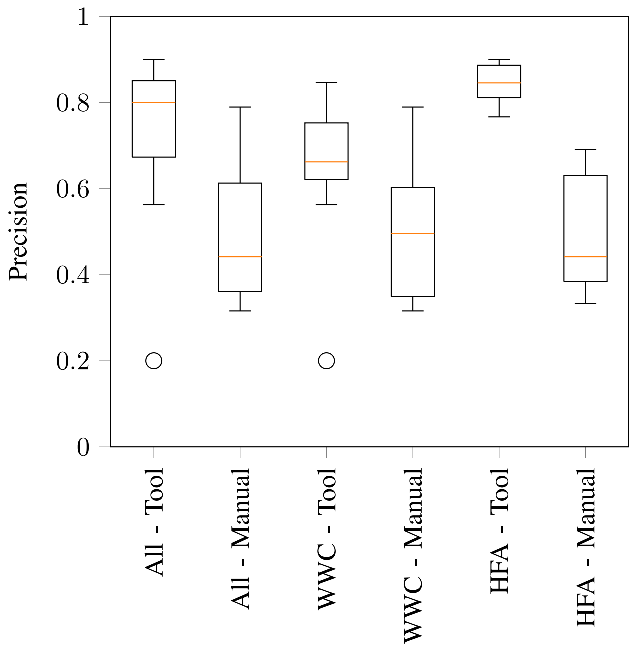

Fig. 2. Precision

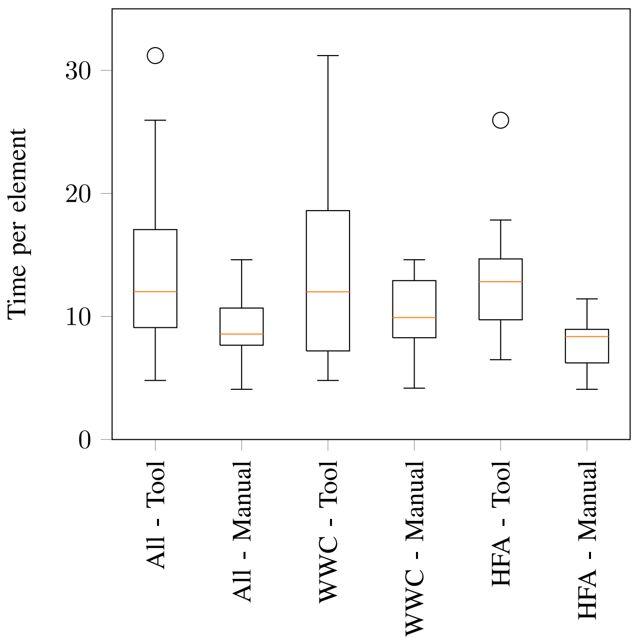

Fig. 3. Time per element

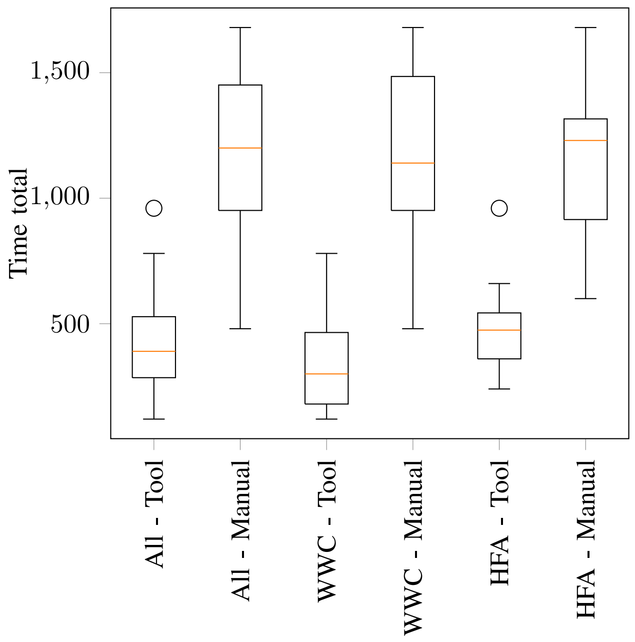

Fig. 4. Total Time

A. Performance Differences Between Groups

For both documents, the achieved recall for finding defects is higher when the tool was used. Averaging over both documents, the recall increases from 0.39 to 0.51. Even though the tool does not return all defects, its usage still resulted in more defects being found. The results for the Hands-Free Access specification are worse than the results for the Wiper Control specification, which may be due to the more complex requirements or the higher number of defects (see Table II). The recall of one participant in the treatment group is particularly low on the Wiper Control specification. This participant made 5 changes, and fixed only 1 out of 19 defects in the document.

Overall, not a single participant in the control group exceeded the highest possible recall of the treatment group (limited by the fact that the tool does not return all defects). This reveals that even though the tool does not allow the user to find all defects, it does not result in worse recall.

The precision increased considerably with tool usage. Precision of the control group averages around 0.5, which means that only half of their edits corrected actual defects; the other half introduced new defects into the document. The control group did not manage to improve the the quality of the specifications. The results of the treatment group are substantially better, precision averages at 0.8.

The time per element is about 10 seconds. This is close to what has been measured during the previous experiment as well. For both specifications, the participants of the treatment group used more time than the participants of the control group. This is in line with our expectations since the participants of the treatment group had to do more work per element as described in Section III-G. Overall, the average time per element increases from 8.9 s to 13.9 s (56% increase). No significant differences can be observed between both specifications. However, a few participants in the treatment group needed much more time than usual.

The total time statistics finally reveal that the treatment group finished reviewing the specifications much faster than the control group, even though they used more time for each element. This was to be expected, since the participants in the treatment group had to inspect only a fraction of the elements.

Overall, the average total time decreases from $1163\,\mathrm{s}$ to $432\,\mathrm{s}$ (63% decrease).The total time needed to review the Hands-Free Access specification is larger than the total time needed to review the Wiper Control specification since it is slightly longer (see again Table II).

Overall, the results reveal that by using the tool in reviews increases precision and recall on defect detection and decreases total time needed by the participants significantly, although more time is needed to make a decision for each element.

B. Calculation of β and Summarization

We can now calculate β for our classification task:

$$\begin{aligned}\beta &= \frac{TimeTotal(CG)}{ErrorsCorrected(CG)} * \frac{1}{TimePerElement(TG)} \\ &= \frac{18\,609\,\mathrm{s}}{216} * \frac{1}{13.94\,\mathrm{s}} \\ &= 6.18 \\ &\approx 6.2\end{aligned}$$

The calculation of β is based on times measured during the experiments. Therefore, β can only be as accurate as the underlying measurements. We used error propagation to determine how accurate β is. Assuming that the each time measurement of the participants has an error of $\pm 30s$, the relative error of β is $\pm 1.8\%$. When working with students, the optimal β for our classification task is within the range 6.0 to 6.3. This value indicates that it is much more important for our classification task to provide a tool with good recall instead of a tool with balanced recall and precision. We are also allowed to make compromises regarding precision.

To estimate summarization, defect detection recall and precision on actual requirements data, we have performed a synthetic test as described in III-G, Calculation of summarization. For this test, the dataset that was used to train the classifier for the experiment was used again. We trained the classifier of the tool with the same settings used for the experiment and tested the classifier with standard 10-fold cross validation. In each of the 10 folds, we used 90% of the data for training and 10% for testing in such a way, that each element in the dataset would be used for testing exactly once.

However, we want to evaluate the ability of the classifier to identify defects. Therefore, we introduced defects into the *test* set of each fold so that 16.5%[1] of all elements are labeled incorrectly. Rather than evaluating whether the classifier can correctly predict the label of an element in the test set, we evaluate whether the classifier is able to detect these defects.

Figure 5 shows the precision and recall for detecting defects and allows us to make the following observations. We cannot achieve both high precision and high recall. If we want a reasonably high recall (0.95), precision is somewhere below 0.5. On the opposite hand, If we want high precision (0.95), recall drops significantly. The graph also shows the summarization.

[1]Amount of defects based on Table II

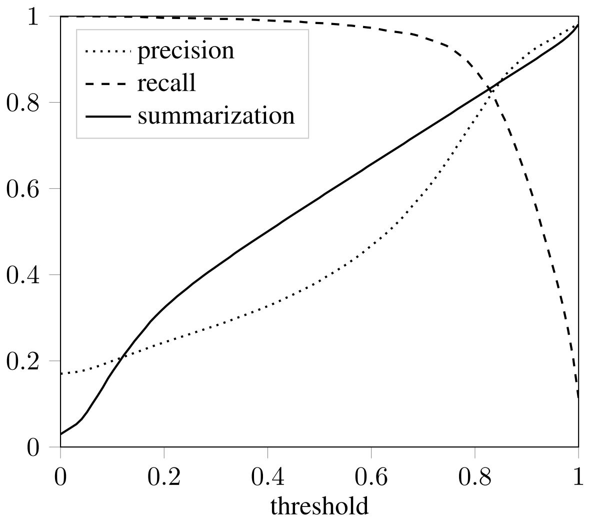

Fig. 5. Precision, recall, and summarization

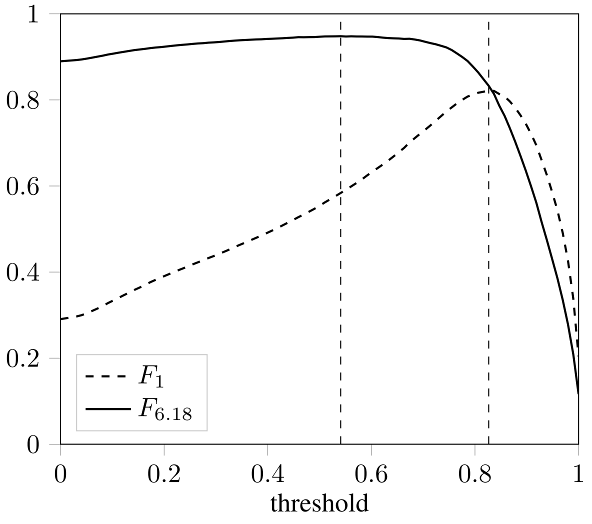

Fig. 6. F_1 and $F_{6.18}$

Even with high recall (0.95), summarization is still greater than 50%, which we consider to be very good.

We used precision and recall to calculate both F_1 and F_β with β set to 6.18. The results are displayed in Figure 6. The vertical lines represent the maximum of F_1 and F_β, respectively. If we use the F_1 measure to optimize our classifier, we set the threshold[2] to 0.83 and have a classifier that has a recall of 0.83, a precision of 0.81 and a summarization of 0.83. However, we need to emphasize recall more and therefore used the F_β score instead. With this score, we optimized the classifier by setting the error detection threshold to 0.54. The classifier now has a recall of 0.98, a precision of 0.42 and a summarization of 0.61. As a result of our empirical experiment, we know that these values represent a good balance between

[2]Thresholds vary with training settings such as epochs and regularization. Therefore, thresholds are only valid for one particular trained model.

recall and precision for the classification task when conducted with students. Increasing recall even more would diminish the benefits from having high summarization. Decreasing recall would reduce the usefulness of our approach because more defects would be hidden from the user.

C. Implications for RE Classification Tasks

When requirements engineers review a specification and search for defects, they analyze each element of the specification and assess it based on certain quality criteria. However, most of the elements may already meet these criteria and therefore do not need any further inspection. Nonetheless, the requirements engineer still uses time do check these elements. By using a tool that reduces the number of elements the requirements engineer has to inspect, they are able to save a considerable amount of time.

In case of the problem of classifying specification elements into requirements and non-requirements, a trained classifier is able to reduce the number of elements to be inspected by 61% on average while still keeping almost all elements with defects (98%) in the returned subset. This will of course vary by specification (i.e., summarization may be worse on specifications of poor quality).

This and previous studies [10] have shown that using tools based on natural language classifiers may be beneficial for the requirements engineer performing the task, because such a tool will reduce the overall time needed to perform the review.

D. Threats to Validity

There are a few aspects of our study that may limit the usefulness of the results. These will be listed below.

First and foremost, students are no requirements engineering experts. They have less knowledge about the documents and therefore may decide differently than requirements experts. Empirical studies with experts may yield different results. Such a study may or may not yield a beta that is different from the one obtained in the study presented in this paper:

- Experts may be faster at performing the task. They may need less time both with and without the tool. Therefore, β should not be significantly different (i.e., close to one or above 10).
- However, experts may also be able to find more defects compared to students. This may be true especially when not using the tool, given the bad performance of the students in the control group. The increased amount of defects found will result in a β smaller than 6.

Furthermore, we did not check for statistical significance because our sample size is too small. Although we can observe repeating patterns in the results of both specifications, repeating the experiment with different students may lead to better or worse results. It is very difficult to perform large-scale evaluations (i.e., with online surveys), since the specifications used for the experiment still contain confidential information and cannot be made public.

Maturation is an effect that occurs over time and may change a subject's behavior due to learning, fatigue, or changes in motivation. In our experiment, the students may have learned something about the given task in the first session and applied that knowledge to the second specification.

All students finished in time. However, the time limit may have forced them to work faster in order to finish within the time limit, resulting in worse results. This may especially affect the performance of the control group, since they had considerably more elements to inspect.

The gold standard used to evaluate the student was set by us and not actual requirements engineers. Since we have worked many years on this and similar classification problems, we consider it to be very close to the actual truth.

V. Related Work

Berry reports that there is empirical evidence that β is greater than 1 for a variety of tasks, and in many cases, significantly so [2]. He calculates $\beta = 18.4$ for a particular tracing task [14].[3] For the task of finding ambiguities in requirements specifications, he calculated $\beta = 8.7$ based on numbers from an evaluation of SREE, an ambiguity finder [16]. For the task of finding feature requests in app reviews, he calculated $\beta = 9.09$, for the task of finding bug reports, he calculated $\beta = 10.00$, for the task of estimating user experiences from app reviews, he calculated $\beta = 2.71$. All three estimates are derived from an evaluation of an app review classification tool [5].

In summary, the determined values for β range from 2.71 up to 18.45. Still, most authors evaluate their approaches based on F_1. Within this range of calculated βs, our determined $\beta = 6.18$ looks reasonable and resides close to related tasks such as identifying bug reports in app reviews.

A. Studies Considering Precision/Recall Imbalances

Recently, more and more authors consider the balance between precision and recall for their problems at hand. A few authors already suspected that recall may be more important than precision and therefore used F_2 to evaluate their approaches [17], [18], [19], [20].

For example, Scandariato et al. [21] deliberately value recall higher than precision and suggest using F_2 as well. The calculations of Berry as well as our results show that even F_2 does not give enough weight to recall.

Rahman et al. [22] and Canfora et al. [23] agree that while broadly applicable, precision and recall by itself is not well-suited for the quality-control settings in which defect prediction models are used. They recommend a combination of both, effectiveness (e.g., precision and recall) and inspection cost, as the decision-making criteria of prediction models.

In other related defect prediction studies, the authors shift towards using more practical performance evaluations [24]. Mende and Koschke [25] proposed bug prediction models that are effort-aware and compared strategies to include the effort treatment into defect prediction models. Following their proposition Kamei et al. [26], evaluate two common defect prediction findings (i.e., process metrics outperform product

[3]The original paper reports a value of 73.6, which was corrected afterwards in a technical report [15].

metrics and package-level predictions outperform file-level predictions) when effort is considered. They find that, when effort is considered, the first finding holds while the second finding does not.

Menzies et al. [27] inspected recent studies and concluded that these were not able to improve defect prediction results. Their explanation includes that performance measured as a trade off between the probability of false alarms and the probability of detection is not enough to justify improvement. They also suggest changing the standard goal to consider effort, i.e., to find the smallest set of modules that contain most of the defects [24].

B. Alternative Evaluation Metrics

Since the F_β-measure is not based on the complete confusion matrix, its usage may be regarded as insufficient [28]. In other works the following two metrics are considered more useful.

The area under the ROC curve (AUC) [29] is used to indicate a model's capability to distinguish between classes. Commonly used to present results for binary decision problems, the AUC provides an aggregate measure of performance across all possible classification thresholds. Therefore AUC is classification threshold invariant. Although a deep connection exists between AUC and Precision-Recall curves, the latter provide a more informative picture of an algorithm's performance [30]. The AUC is hard to interpret since relative costs of False Positives and False Negatives are usually not provided [31].

Shepperd et al. [31] advocate *the Matthews correlation coefficient (MCC)*. The MCC takes true and false for each, positives and negatives into account and is considered a balanced measure. It can be used even if the classes are of very different sizes, a common property of software defect data [31], [32].

VI. Conclusion

In this paper, we have evaluated the ability of tools to assist requirements engineers in a specific requirements engineering task. This task involves deciding for each element of a specification whether it is a requirement or a non-requirement. The tool assist requirements engineers in reducing the number of elements they need to inspect by hiding elements which are most likely correctly labeled. This is accomplished by using neural networks. We evaluated this approach by performing a controlled experiment with students. The results were then used to determine the optimal balance between recall and precision for our task.

Overall, the results look very promising. We will now summarize the findings of this paper regarding our research questions.

- **RQ 1:** *Is there any performance difference between the two groups?* When supported by the tool, our participants performance measured in defect detection recall and precision increased. The participants were also considerably faster due to the reduced amount of elements to inspect, even though they needed more time to review each element.
- **RQ2**: *What is the optimal β for the requirement/non-requirement classification task?* As calculated by the results of the experiment, a good estimate of β when working with students on this classification task is 6.2. When working with experts, β will probably be close to 6.
- **RQ3**: *How big is summarization given β on typical requirements specification documents?* Without tuning the classifier towards either precision or recall, summarization is about 76% on both our test documents. When we use β to weight recall more, summarization on typical requirements specifications is about 61%. Therefore, requirements experts can save a considerable amount of time by using the tool.

Other classification tasks or even general requirements engineering tasks might also benefit from proper β-optimization of recall and precision. This includes many quality control tasks (i.e., find ambiguous, duplicate and incomplete requirements) and link detection tasks (i.e., top-down-traceability). When it is possible to create a classifier for any given task and the classifier achieves reasonable accuracy, using a tool to assist in this specific task is worth investigating.

The tool can not only be used to assist reviews but maybe other purposes as well. The classifier might be able to create an initial labeling of specification elements as they are written by requirements engineering experts. Such a tool may automatically set the label on new elements when classification confidence is high and ask the requirements engineer when the confidence is low. Such a tool may also be used to train new employees who are still new to this classification task. Overall, even though the approach presented in this paper is imperfect regarding the recall, the provided benefits outweigh its deficits.

References

[1] A. Ferrari, F. Dell'Orletta, A. Esuli, V. Gervasi, and S. Gnesi, "Natural language requirements processing: A 4D vision," *IEEE Software*, vol. 34, no. 6, pp. 28–35, 2017.

[2] D. M. Berry, "Evaluation of tools for hairy requirements and software engineering tasks," in *25th IEEE International Requirements Engineering Conference Workshops (REW)*, 2017, pp. 284–291.

[3] J. P. Winkler and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks," in *3rd IEEE International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, 2016, pp. 39–45.

[4] Z. Kurtanović and W. Maalej, "Automatically classifying functional and non-functional requirements using supervised machine learning," in *25th IEEE International Requirements Engineering Conference (RE)*, 2017, pp. 490–495.

[5] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Requirements Engineering*, vol. 21, no. 3, pp. 311–331, 2016.

[6] D. Ott, "Automatic requirement categorization of large natural language specifications at mercedes-benz for review improvements," in *Requirements Engineering: Foundation for Software Quality (REFSQ)*, J. Doerr and A. L. Opdahl, Eds. Springer Berlin Heidelberg, 2013, pp. 50–64.

[7] C. C. Aggarwal and C. Zhai, "A survey of text classification algorithms," in *Mining Text Data*, C. C. Aggarwal and C. Zhai, Eds. Springer US, 2012, pp. 163–222.

[8] Y. Kim, "Convolutional Neural Networks for Sentence Classification," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1746–1751.

[9] J. P. Winkler and A. Vogelsang, “What does my classifier learn? A visual approach to understanding natural language text classifiers,” in *22nd International Conference on Natural Language & Information Systems (NLDB)*, 2017, pp. 468–179.

[10] ——, “Using tools to assist identification of non-requirements in requirements specifications – a controlled experiment,” in *24th International Working Conference Requirements Engineering: Foundation for Software Quality (REFSQ)*, 2018.

[11] A. J. Ko, T. D. LaToza, and M. M. Burnett, “A practical guide to controlled experiments of software engineering tools with human participants,” *Empirical Software Engineering*, vol. 20, no. 1, pp. 110–141, 2015.

[12] A. Jedlitschka, M. Ciolkowski, and D. Pfahl, “Reporting experiments in software engineering,” in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjøberg, Eds. Springer London, 2008, pp. 201–228.

[13] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer Science & Business Media, 2012.

[14] T. Merten, D. Krämer, B. Mager, P. Schell, S. Bürsner, and B. Paech, “Do information retrieval algorithms for automated traceability perform effectively on issue tracking system data?” in *Requirements Engineering: Foundation for Software Quality (REFSQ)*, M. Daneva and O. Pastor, Eds. Cham: Springer International Publishing, 2016, pp. 45–62.

[15] D. M. Berry, “Evaluation of tools for hairy requirements engineering and software engineering tasks,” School of Computer Science, University of Waterloo, Tech. Rep., 2017. [Online]. Available: https://cs.uwaterloo.ca/~dberry/FTP_SITE/tech.reports/EvalPaper.pdf

[16] S. F. Tjong and D. M. Berry, “The design of SREE — A prototype potential ambiguity finder for requirements specifications and lessons learned,” in *Requirements Engineering: Foundation for Software Quality (REFSQ)*, J. Doerr and A. L. Opdahl, Eds. Springer Berlin Heidelberg, 2013, pp. 80–95.

[17] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, “A machine learning approach for tracing regulatory codes to product specific requirements,” in *32nd ACM/IEEE International Conference on Software Engineering (ICSE)*, 2010, pp. 155–164.

[18] H. Yang, A. de Roeck, V. Gervasi, A. Willis, and B. Nuseibeh, “Analysing anaphoric ambiguity in natural language requirements,” *Requirements Engineering*, vol. 16, no. 3, 2011.

[19] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, “Automated checking of conformance to requirements templates using natural language processing,” *IEEE Transactions on Software Engineering*, vol. 41, no. 10, pp. 944–968, 2015.

[20] A. Delater and B. Paech, “Tracing requirements and source code during software development: An empirical study,” in *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ICSE)*, 2013, pp. 25–34.

[21] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, “Predicting vulnerable software components via text mining,” *IEEE Transactions on Software Engineering*, vol. 40, pp. 993–1006, 10 2014.

[22] F. Rahman, D. Posnett, and P. Devanbu, “Recalling the "imprecision" of cross-project defect prediction,” in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE ’12. New York, NY, USA: ACM, 2012, pp. 61:1–61:11. [Online]. Available: http://doi.acm.org/10.1145/2393596.2393669

[23] G. Canfora, A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, “Defect prediction as a multiobjective optimization problem,” *Softw. Test. Verif. Reliab.*, vol. 25, no. 4, pp. 426–459, Jun. 2015. [Online]. Available: https://doi.org/10.1002/stvr.1570

[24] Y. Kamei and E. Shihab, “Defect prediction: Accomplishments and future challenges,” in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 5, March 2016, pp. 33–45.

[25] T. Mende and R. Koschke, “Effort-aware defect prediction models,” in *2010 14th European Conference on Software Maintenance and Reengineering*, March 2010, pp. 107–116.

[26] Y. Kamei, S. Matsumoto, A. Monden, K. Matsumoto, B. Adams, and A. E. Hassan, “Revisiting common bug prediction findings using effort-aware models,” in *2010 IEEE International Conference on Software Maintenance*, Sep. 2010, pp. 1–10.

[27] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, “Defect prediction from static code features: current results, limitations, new approaches,” *Automated Software Engineering*, vol. 17, no. 4, pp. 375–407, Dec 2010. [Online]. Available: https://doi.org/10.1007/s10515-010-0069-5

[28] D. M. W, “Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation.”

[29] A. P. Bradley, “The use of the area under the roc curve in the evaluation of machine learning algorithms,” *Pattern Recognition*, vol. 30, no. 7, pp. 1145 – 1159, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320396001422

[30] J. Davis and M. Goadrich, “The relationship between precision-recall and roc curves,” in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML ’06. New York, NY, USA: ACM, 2006, pp. 233–240. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143874

[31] M. Shepperd, D. Bowes, and T. Hall, “Researcher bias: The use of machine learning in software defect prediction,” *IEEE Transactions on Software Engineering*, vol. 40, no. 6, pp. 603–616, June 2014.

[32] S. Boughorbel, F. Jarray, and M. El-Anbari, “Optimal classifier for imbalanced data using matthews correlation coefficient metric,” *PLOS ONE*, vol. 12, p. e0177678, 06 2017.