2018 IEEE 11th International Conference on Software Testing, Verification and Validation

# Investigating NLP-based Approaches for Predicting Manual Test Case Failure

Fatemeh Sharifi
Department of Electrical and Computer Engineering
University of Calgary
Calgary, Canada
fatemeh.sharifi1@ucalgary.ca

Hadi Hemmati
Department of Electrical and Computer Engineering
University of Calgary
Calgary, Canada
hadi.hemmati@ucalgary.ca

***Abstract*—System-level manual acceptance testing is one of the most expensive testing activities. In manual testing, typically, a human tester is given an instruction to follow on the software. The results as "passed" or "failed" will be recorded by the tester, according to the instructions. Since this is a labour-intensive task, any attempt in reducing the amount of this type of expensive testing is essential, in practice. Unfortunately, most of the existing heuristics for reducing test executions (e.g., test selection, prioritization, and reduction) are either based on source code or specification of the software under test, which are typically not being accessed during manual acceptance testing.**

**In this paper, we propose a test case failure prediction approach for manual testing that can be used as a non-code/specifcation-based heuristic for test selection, prioritization, and reduction. The approach uses basic Information Retrieval (IR) methods on the test case descriptions, written in natural language. The IR-based measure is based on the frequency of terms in the manual test scripts. We show that a simple linear regression model using the extracted natural language/IR-based feature together with a typical history-based feature (previous test execution results) can accurately predict the test cases' failure in new releases. We have conducted an extensive empirical study on manual test suites of 41 releases of Mozilla Firefox over three projects (Mobile, Tablet, Desktop). Our comparison of several proposed approaches for predicting failure shows that a) we can accurately predict the test case failure and b) the NLP-based feature can improve the prediction models.**

***Index Terms*—NLP; POS; TF/IDF; Neural Network; Linear Regression; Test Failure Prediction; Manual acceptance testing;**

## I. Introduction

Software testing is a time-consuming and costly activity in the software development life cycle. Automation is a promising solution to decrease the cost of testing and increase the quality of the testing process. Test automation can be applied on several testing problems such as test data/case generation, test prioritization, test execution, and test repair. Most of the existing test automation techniques rely on metrics that are extracted either from source code (e.g., code coverage [1]) or specification models (e.g., UML state machines [2], [1]) of the software under tests (SUT). However, a large body of testing in practice is done manually. Such test cases are written in natural languages. They describe a scenario to follow on the SUT. A typical example of such manual testing is system acceptance testing were the ready-to-ship features are being verified by the manual testers.

Manual testing is specifically interesting, from the automation point of view, because it is one of the most expensive type of testing that requires a human to judge the pass/fail results for every single run of the test cases. This might be still manageable for small systems with limited number of manual test cases, but when the system size grows, running all manual test cases for every feature change will not be possible within the time limits. The time limitation becomes more serious when the development team follows a rapid release (continuous delivery) practice.

"Test case failure prediction" refers to approaches that estimate the likelihood of a test case execution failure. These predictions then can be used in deciding what portion of existing test cases to run if we have a limited budget (test selection). Or in which order should we run test cases to catch defects sooner (test prioritization). Most of test prediction research studies rely on various traditional software metrics, on the code or specification-level, e.g., LOC, complexity, and churn [3], [4]. However, in the context of manual acceptance testing, typically, these artifacts are either not accessible or not linkable to the manual test cases (e.g., a manual acceptance test instruction is not easily traceable to the underling source code elements). Therefore, in this work we are exploring other measures for test case failure prediction that do not require software's source code or specification/requirement documents.

One of the best measures that do not need any source code or specification/requirement documents is historical execution result of a test case. There are many studies [5], [6], specially in the context of regression testing, which demonstrate that the previous failure of a test case is a very good indicator of its future failure. For example, history-based techniques have been shown to outperform text analytics-based approaches for prioritizing manual test cases, in the past [7], [8].

In this paper, we introduce a novel approach for manual test case failure prediction, based on Natural Language Processing (NLP), that improves accuracy of the traditional history-based predictions. As discussed, the test cases in manual testing are written in a natural language. Therefore, the objective of the new metric is extracting some insights (features keywords) from the test scripts. Using these feature vectors, as inputs of prediction models, helps improving the accuracy of traditional

0-7695-6388-0/18/$31.00 ©2018 IEEE
DOI 10.1109/ICST.2018.00038

history-based approaches, where each test case is represented just by an ID (no feature is mined). Our proposed features are mined using a simple NLP technique, called Part of Speech Tagging (POS) [9], to extract keywords in test cases and then weighting them using a Term Frequency Inverse Document Frequency (TF-IDF) [10] metric.

We have conducted an extensive experiment with manual test cases of Mozilla Firefox's Desktop, Mobile, and Tablet projects with 13, 14 and 14 releases, respectively. The results show that our proposed NLP-based model leads to up to $24\%$ (on average $4\%$) higher accuracy compared to the baseline (traditional history-based approach).

The contributions of this paper are as follows:

- Using an NLP technique (POS) to extract features to represent manual test cases.
- Using an IR metric, TF-IDF, to build a prediction measure, based on the extracted features.
- Empirically investigating the effects of the NLP/IR-based features, the historical data, and the prediction models on the predictions accuracy.
- Conducting a large-scale empirical study on a real-word software system with over 23,085 test cases and 2,457 real bugs.

The rest of this paper is organized as follows: Section II describes the background in test case prediction, NLP and IR methods, and prediction models; our proposed NLP-based methods for test case prediction has been presented in Section III. We have explained our experiments and results in Section IV and finally, Section V concludes the paper and summarizes our future work.

## II. Background and Related Work

In this section we cover backgrounds on test case prediction including the basics of neural network, linear, and nonlinear regression techniques. In addition, the NLP and IR methods that are being used in this study will be explained here.

### A. Test Case Prediction

There has been a lot of studies in recent year about defect prediction. To predict a defect, one may use product (e.g., source code, test code), process, and organizational metrics, using several mining software repositories approaches [3]. Testing-related metrics had been also successfully applied on the defect prediction problem [11]. However, in this paper, we are not interested in defect prediction and rather want to predict the test case failure. The main difference is that we are focusing on test case-related features as opposed to code-base features (such as LOC, code complexity, and code change), assuming that the code-base may not be available. Common testing features include the size of a test case, code and requirement coverage, and historical test case execution results (pass and fail) [12]. Note that even though, our focus is on test case failure prediction but most of the underlying techniques and approaches are inspired by the defect prediction studies.

Test case prediction techniques can be studied from two perspectives: a) what features or metrics it has access to (available data) and b) what predictive model it uses for estimating the test case failure?

*1) Features in test case prediction:* In general, any product, process, or organizational feature that is used in defect prediction can also be used in test case failure prediction. Below we list some of the common features that have been used for test case failure prediction in past.

*a) Code coverage:* In the test failure prediction domain, code coverage is the common measure of quality for test cases. For instance, different coverage-based approaches have been used for test case prioritization. However, such approaches assume that we have access to the code coverage information of test cases, either from the previous executions of the test case [6] or by static analysis [13]. In this study, we are focusing on features that can be extracted only using the test cases in a manual testing settings on the system level. Therefore, code-based coverage info is not available.

*b) Historical results:* Another typical test case failure prediction feature, in practice, is the previous execution results per test case. The assumption here is that the historical results of test case executions are recorded and accessible for the prediction method. Test cases that have detected faults in the past may be regarded as "proven performers" [14], hence should be given more probability to fail again. Therefore, a simple history-based metric would be whether a test case has ever failed in the past or not. The positive answers will be predicted as failing tests. This idea is typically used for regression testing [5]. Most recently, Noor et.al. proposed a history-based prioritization approach where the similarity of modified or new test cases to old failing test cases have been considered as input features for prediction [6], [12]. In this study, we will use history-based features as baselines of comparison.

*c) Text-based features:* Text-based features are a newly introduced category that can work with even the most limited datasets with no source code, coverage, or execution information. Basically the only requirement is the textual representation of the test case (can be in programming/scripting/natural languages) [15]. For instance, in [7], the authors have used a topic modeling technique called LDA for representing manual test cases. In this paper, we also introduce novel text-based metrics.

*d) GUI coverage:* In the domian of event-driven systems such as Web and mobile applications, modeling test cases based on their sequence of events is a common approach in test generation and prioritization. For instance, Bryce et. al. [16], [17] used the notion of t-way event interaction [18] coverage as a heuristic for test prioritization, where the goal is covering all t-way event interactions, as soon as possible. In another work, Sampath et. al. prioritize web applications's test suites by test lengths, frequency of appearance of request sequences (which are the "events" in this context), and systematic coverage of parameter-values and their interactions [19]. Since many manual testing instructions are based on GUI menus, one can potentially extract such event sequences from manual test cases and use them as a prediction feature. However, we did


Fig. 1. Overview of our approach

not use this approach in this study, because we did not want to be limited to manual testing based on GUI menus. In addition, we did not have access to the GUI of all previous versions of the software under tests, in this study.

*2) Test case prediction models:* Classifiers such as Linear and Non-Linear regression (LR and NLR) and machine learning models such as Neural Network (NN) and Support Vector Machine (SVM) have been extensively used in bug triage and prediction studies [20], [21], [22]. In this study, we use the same insights and apply LR, NLR and NN, as the most common techniques, to learn a classification model from the extracted features per test case. We then use that model to predict future test cases' failures based on the current attribute values.

*a) Linear Regression:* A linear regression (LR) method is one of the most common approaches in statistics and it works in application of numeric prediction especially where both the output class and the features are numeric [23]. Basically, the output class is expressed as a linear combination of the attributes with their corresponding weights. The LR function is given by:

$$\mathbf{y} = b_0 + \sum_{i=1}^{n} b_i x_i \quad (1)$$

where $y$ is the output, $x_i$ is the input variable, $b_i$ is the regression coefficient of explanatory variable $i$, and $b_0$ is the value of the intercept in the linear fitting.

*b) Nonlinear Regression:* The nature of the input data versus the output may suggest there is a nonlinear relationship. The simplest way to try to estimate such a relationship is through a quadratic regression model which has been used in this paper. While LR gives a clear analysis of the relationship between output and each single input, quadratic regression takes the interactions between input variables into account [5], thereby, making it a better fit for nonlinear systems. Quadratic regression is generated from LR by adding more terms to equation 2. The quadratic regression function is:

$$\mathbf{y} = b_0 + \sum_{i=1}^{n} b_i x_i + \sum_{\substack{1 \le i \le n-1 \\ i+1 \le j < n}} b_{ij} x_i x_j + \sum_{i=1}^{n} b_{ii} x_i^2 \quad (2)$$

with same definition as per equation 2.

*c) Artificial Neural Networks:* Artificial Neural Networks (ANNs) have been employed in many domains because of their strength in classification and data pattern recognition. ANNs can be defined as an interconnected group of simple processing neurons, of which the functionality is based on the human brain neurons [23]. The processing ability of the network is stored in the weights of connected neurons that are obtained and adjusted by a learning process from a set of training patterns. A trained ANN is capable of classifying non-linear and multidimensional input-output patterns.

A simple neuron receives signals from *n* inputs, each of which has its own connection weights. The neuron subsequently evaluates the signals by adding the product of each input and the connection weights associated and comparing the summation to its threshold value. The vector calculated is then converted using the transfer function F. Consequently, the converted value is the output of the neuron.

### *B. Natural Language Processing*

*1) Overview:* Natural Language Processing (NLP) techniques, which can extract structured information from free text, can be very useful in software automation when the input data is text or speech. Parsing natural languages is one of the most basic requirements for automating any system that depends on the input from human in the form of text or speech. Natural Language Processing (NLP) is a popular topic in Computational Linguistic which deals with interfacing computer with human languages. NLP specifically deals with extracting context out of natural language text by using grammar inference, relating the words with each other and using this information to address natural language semantics [24]. There are several sub-problems in NLP one of which is POS tagging (assigning part of speech tags to words), which is being used in this paper.

*2) Part of Speech (POS) Tagging:* Part-of-Speech (PoS) tagging [25], [26] is the task of labeling each word in a sentence with its appropriate syntactic category, called part of speech. Part-of-speech (POS) taggers, identify POS of a word and tag it as a noun, verb, preposition, etc. and then parse the tagged words into grammatical phrases to help distinguish the semantics of the component words. This technique is a very important preprocessing task for almost all systems with the input from human speech or text. POS tagging has received great attention from software engineering researchers developing text-based software engineering tools [27].

In the context of software engineering, NLP is not a new topic and has been used for decades. Main applications of NLP in software engineering are in the areas of Specification Mining and Requirement Engineering. In 1989, Saeki et al. [28] used NLP techniques to derive formal specifications from informal ones using similar idea as presented in this paper; extracting nouns and verbs as representations of classes, attributes and methods in the code base. Hudaib et al. [2] presented a two-way approach using NLP techniques; to

extract UML state machines from English natural language requirements, and then back from UML model to English natural language requirements.

*3) Term Frequency-Inverse Document Frequency (TF-IDF):* TF-IDF is a very common metric in the context of Text Retrieval. It is a measure that is assigned to a term in a corpus of documents. The first part, Term Frequency (TF), simply uses the term's frequency to record the number of times that it appears in a document normalized by the total number of terms in that document:

$$tf(t,d) = \frac{count(t,d)}{\sum_{v \in d} count(v,d)} \tag{3}$$

IDF which is the second part of TF-IDF measures how rare or common this term is across all documents in the corpus. It is calculated by dividing the total number of documents (D) by the number of documents that contain that term (df). IDF is typically presented as the logarithmic version of the above calculation.

$$idf(t) = log \frac{|D|}{|\{df\}|} \tag{4}$$

The TF-IDF of a term t is simply t's tf multiplied by its inverse document frequency.

$$tfidf(t,d) = tf(t,d) \cdot idf(t) \tag{5}$$

where,

$tf$ is the term frequency.
$df$ is the document frequency, i.e., the number of documents containing the word.
$D$ is the number of documents.

In this research, TF-IDF is used to measure the strength of features extracted from POS tagging.

## III. Approach

In this section, we explain our NLP-based approach for test case failure prediction, as per Fig. 1, by first encoding the test cases using the mined features and then applying the prediction technique on the encoded test cases.

### A. Test Case Encoding

A manual test case is typically a document that details the sequence of actions that testers need to take and the outputs they are expected to observe. A sample test case (only the instructions, not the expected outputs) is shown in Table I from Mozilla Firefox Mobile v16 (which also appears in v17, v18 and v27). In the absence of code and specification, our encoding's objective is to represent each test case as a set of keywords from the textual instruction. The keywords are supposed to give insights about the software features under test.

In this study, we use the set of nouns in the test instruction as our representation. One reason behind choosing nouns is that when manual test instructions refer to user interface elements, including the menus and sub-menus and their states that all need to be tested, they are all nouns. In addition, for cases where the manual test instruction is not directly refers to the user interfere, nouns can still provide a rich summary of the instruction.

As an example, for the test case that is represented in Table I the list of nouns are <fennec, portrait, mode, device, landscape, mode, device, portrait, mode>. As it can be seen, all nouns except fennec, which refers to the browser itself, are valid estimations of either UI elements or concepts that are being tested. Intuitively, we can guess that this estimation approach has a very high recall, since when a test case verifies a GUI element or a software feature, it's very likely that the test instruction directly uses the element's or feature's name, as a noun. However, the precision of the heuristic is not clear and there might be some noises in the nouns as well. Therefore, one of the factors of the final results is the choice of heuristic.

To automatically exclude noises as much as possible, in our experiment, we first preprocess all the test cases by excluding all URLs, special characters and numbers; typical preprocessing in standard text-mining [29]. Further, we split words, based on camel case and underscore, remove stop words, and stem multiple forms of the same word in one form. We use the Stanford Maximum Entropy Tagger [9] to extract nouns from the entire test suite. We ignore the sub-classifications of nouns and keep only words that are tagged as any sort of noun. Table II shows the tagging results for the sample test case of Table I. Finally, each test case is represented as a set of nouns that are identified by the POS tagger.

### B. NLP-based Prediction models

Our prediction models are LR, NLR and NN with two features. The first feature is based on the traditional history-based feature and the second feature is the "noun" coverage of the test cases. To calculate the above features we follow the following steps:

**Step 1:** Calculate the history-based measures

- Simple History (SH): Assign value ONE to the test cases of this version if they passed in the immediate previous version assign value ZERO to the previously failed or new test cases.
- All History (AH): Assign the average of SH values in all previous versions. This measure considers the entire history rather than just the immediate version (SH).
- Weighted History (WH): Assign weighted average (as opposed to simple average of AH measure) of SH values in all previous versions with more importance on recent versions (decremental weights when starting from the newest release).

**Step 2:** Calculate the "noun" coverage measure

- Tag all words in all test cases using a POS tagger and keep only the nouns.
- Represent each test case using its nouns. We call each of these nouns as the *unit* of representations.
- Traverse the entire test suite to find unique units.
- Calculate TF-IDF of each unit per test case.

TABLE I
A REAL TEST CASE FROM MOZILA FIREFOX MOBILE VERSION 16

| Test Case ID:1206, Product: Mobile, Version:16 |
|---|
| **Steps to Perform** |
| 1. Launch fennec while in portrait mode. |
| 2. Rotate the device to landscape mode. |
| 3. Rotate the device back to portrait mode. |

TABLE II
REAL TAGGED TEST CASE FROM MOZILA FIREFOX MOBILE VERSION 16

| Test Case ID:1206, Product: Mobile, Version:16 |
|---|
| **Tagged Steps** |
| 1. fennec/NN portrait/NN mode/NN. |
| 2. device/NN landscape/NN mode/NN. |
| 3. device/NN portrait/NN mode/NN. |

- Assign a measure to each test case which is the sum of TF-IDF of all unit's of the test case divided by the number of units in that test case.

Note that the sum of TF-IDFs basically looks at the coverage of the test as a whole without focusing on individual units.

To have a more clear idea about the process of the TF-IDF measure, let's see a sample test suite. Assume we have four test cases. The algorithm first tags all the test cases in a test suite and represents the test cases in the form of vectors of nouns, as below.

$$Test1 = <fennec, portrait, mode, device, landscape, mode, device, portrait, mode>$$
$$Test2 = <news, article>$$
$$Test3 = <menu, settings, desktop, firefox, preferences, code, fennec, matrix, desktop>$$
$$Test4 = <menu, settings, firefox>$$

We then traverse through the test cases and compute the TF-IDF of all unique nouns per test case as in Table III. Next, we calculate the sum of the TF-IDF for all the nouns in each test case and then divide it by the number of nouns in that test case. For example, feature vector and normalized TF-IDF of the first test case described in Section III are as below:

$$Test1 = <fennec, portrait, mode, device, landscape, mode, device, portrait, mode>$$

Test1 Normalized-TF-IDF:
(0.03+0.13+0.2+0.13+0.06+0.2+0.13+0.13+0.2)/(9+1)=0.121

Finally, we use LR, NLR and NN algorithms to predict the PASS or FAIL for the test cases of each release using the history-based and the TF-IDF measures, as explained. To train our models, these two measures should be calculated for each test case on a learning set (on all releases before the release under study). For each test case in the learning set, we assign a PASS or FAIL label, based on the actual test execution results in that release. The trained model then will be used in the the current release, to predict test failure using the current TF-IDF and history-based measures, per test case.

In the next section, we will apply this approach on real-world test suites.

TABLE III
NOUN TF-IDF FOR TEST 1

| Noun | TF-IDF |
|---|---|
| fennec | 0.03 |
| portrait | 0.13 |
| mode | 0.2 |
| device | 0.13 |
| landscape | 0.06 |

TABLE IV
SYSTEMS UNDER TEST CHARACTERISTICS: MOZILLA FIREFOX DESKTOP TRADITIONAL (TR) AND RAPID RELEASE (RR), MOBILE AND TABLET.

| Type | Release | Date | #Tests | #Faults | Failure rate |
|---|---|---|---|---|---|
| Desktop-TR | 3.0 | Dec-06 | 580 | 127 | 21.90% |
| | 3.5 | Jul-07 | 766 | 138 | 18.02% |
| | 3.6 | 8/2009 | 828 | 88 | 10.63% |
| | 4.0 | 2/2010 | 997 | 150 | 15.05% |
| Desktop-RR | 5.0 | 4/2011 | 1055 | 6 | 0.57% |
| | 6.0 | 4/2011 | 1119 | 4 | 0.36% |
| | 7.0 | 5/2011 | 1111 | 4 | 0.36% |
| | 8.0 | 7/2011 | 1119 | 7 | 0.63% |
| | 9.0 | 8/2011 | 1114 | 4 | 0.36% |
| | 10.0 | 9/2011 | 1108 | 12 | 1.08% |
| | 11.0 | 11/2011 | 1121 | 3 | 0.27% |
| | 12.0 | 12/2011 | 1121 | 2 | 0.18% |
| | 13.0 | 2/2012 | 1189 | 4 | 0.34% |
| Mobile | 16 | Jun-12 | 364 | 79 | 21.70% |
| | 17 | Aug-12 | 367 | 72 | 19.62% |
| | 18 | Aug-12 | 366 | 87 | 23.77% |
| | 19 | Oct-12 | 388 | 76 | 19.59% |
| | 20 | Nov-12 | 381 | 95 | 24.93% |
| | 21 | Jan-13 | 389 | 72 | 18.51% |
| | 22 | Feb-13 | 423 | 79 | 18.68% |
| | 23 | Apr-13 | 426 | 76 | 17.84% |
| | 24 | May-13 | 476 | 82 | 17.23% |
| | 25 | Jul-13 | 514 | 83 | 16.15% |
| | 26 | Aug-13 | 343 | 50 | 14.58% |
| | 27 | Sep-13 | 434 | 84 | 19.35% |
| | 28 | Nov-13 | 380 | 28 | 7.37% |
| | 29 | Dec-13 | 293 | 21 | 7.17% |
| Tablet | 16 | Jul-12 | 368 | 77 | 20.92% |
| | 17 | Aug-12 | 372 | 77 | 20.70% |
| | 18 | Sep-12 | 377 | 98 | 25.99% |
| | 19 | Oct-12 | 385 | 76 | 19.74% |
| | 20 | Nov-12 | 391 | 62 | 15.86% |
| | 21 | Jan-13 | 390 | 67 | 17.18% |
| | 22 | Feb-13 | 396 | 74 | 18.69% |
| | 23 | Apr-13 | 446 | 88 | 19.73% |
| | 24 | May-13 | 448 | 42 | 9.38% |
| | 25 | Jul-13 | 490 | 82 | 16.73% |
| | 26 | Aug-13 | 333 | 50 | 15.06% |
| | 27 | Sep-13 | 372 | 61 | 16.40% |
| | 28 | Nov-13 | 345 | 39 | 11.30% |
| | 29 | Dec-13 | 319 | 31 | 9.72% |

## IV. EMPIRICAL STUDY

In this section, we explain our evaluation of the proposed test case prediction techniques, in the context of an experi-

ment.

### A. Experiment Design

*1) Objective:* The objective of this experiment is to evaluate the effectiveness of our proposed NLP-based test case prediction techniques, in terms of prediction accuracy. The goal is to improve the state-of-practice (traditional history-based approach) for test case failure prediction.

*2) Research Questions:* We have formulated the objective of this study as the following research questions:

**RQ1: Can a basic NLP-based test case prediction method improve the effectiveness of a traditional history-based test case prediction method?** In this research question, we compare the prediction accuracy of our NLP-based Linear Regression method with a simple history-based measure.

**RQ2: Does looking at the entire history, when collecting the history-based measure, improve the prediction accuracy?** In this research question, we upgrade the simple history-based measure (SH), which assumes the history only consists of the last version, to use the entire history (all previous versions). The two proposed new measures are All History (AH) and Weighted History (WH). AH considers an equal importance to the entire history but WH assigns more weight to more recent versions. This research question is divided into two sub-questions to evaluate SH, AH, and WH on the baseline and Linear Regression approach, separately.

**RQ3: Among basic prediction approaches which one provides higher accuracy, when used with the NLP-based features, for test case failure prediction?** In this research question, we investigate the effect of the prediction approach, on test case failure prediction. We use three basic and simple models for this comparison: Linear Regression (LR), Nonlinear Regression (NLR), and Neural Network (NN).

*3) Dataset:* In this paper, we use a data set of 3 Mozilla Firefox projects (Desktop, Mobile and Tablet). The data for these projects are collected from two previously published work on Mozilla Firefox's manual test suites [7], [8]. The Desktop test suites were collected from the test cases of versions 5 to 13 of Firefox Desktop and their execution results. This project was managed by a web-based system known as Litmus. The Mobile and Tablet test suite are collected from a test management tool called Moztrap. These are test cases for versions 16 to 29 of Firefox Mobile and Tablet. Except in the first four versions of Desktop, the rest of releases follow rapid release strategy. The implication of that in our study is that the releases are small with limited changes and failure per release. There are also a lot of overlap in test suites of consecutive versions.

Table IV shows the characteristic of the 13 (four traditional and nine rapid) releases of Desktop Firefox, 14 releases of Mobile Firefox and 14 releases of Tablet Firefox.

In both Litmus and Moztrap, there are records of each test execution. Each test has been executed several times by perhaps different testers. To assign a PASS or FAIL label to test cases, we first exclude unsuccessful executions of tests that are labeled as erroneous test, no PASS or no FAIL in the test report. After this cleaning step, a test case is considered as PASS only if all valid executions of the test, in that version, were passed. Therefore, a test case is considered as FAILED if there is at least one failed execution, assigned to that test case, in that version. The test execution reports also identify the bug *id* which let us identify unique faults and assign each test case to the unique fault(s) that it can detect.

In our study, the results are reported starting from the third version per project. This is due to the fact that each prediction requires a training dataset (previous version) and the history measures per test case in the training set themselves need at least one release, as history. Thus the first version to predict the result for will be the third release per project.

*4) Comparison Baseline:* Our baseline in this study is a simple history-based method (called SimpleHistoryBaseLine or SH-BL, from now on). Basically, we predict the test cases that fail in the previous version to fail again and those that passed previously to pass again. We also predict the new tests (does not exist in the previous version) as fail, to give them a chance to be executed, in the new release. Note that in our dataset, percentages of new test cases from a given release to the next one is typically very low (ranges from 0.3% to 13% in Tablet – 1% to 13% in Mobile – 0.1% to 6% in Desktop Rapid Release, but 12% to 29% in Desktop Traditional release). The SH-BL prediction method is the foundation of many test case prioritization approaches. Despite its simplicity this approach works very well in regression testing, specially when the releases are frequent (the changes are limited in each iteration).

Another technique that could be considered here as a baseline of comparison was an alternative text mining-based approach for extracting features from the textual test cases. From the existing literature, the only such approach is an LDA-based test failure prediction, which has shown poor results compared to SH-BL [7], [8], [15]. Thus we do not consider that as a baseline.

There are also some techniques which are based on GUI coverage [17], [19] or image understanding from the GUI [30], which could only be applicable if our dataset would include GUI screen-shots and images per test cases for all versions of the history. Therefore, we keep SH-BL as our baseline of comparison .

*5) Normalized-TF-IDF:* To calculate the TF-IDF measure, we divide the raw TF-IDFs by the sum of TF-IDFs of all nouns to normalize the measure, and we call this measure "Normalized-TF-IDF", which will be used as one of the two inputs to the prediction models. In addition, in order to avoid a "Division by zero" in test cases without any noun, we added one to the size of the nouns.

*6) Neural Network Set up:* The NN we employed in this study is a simple multi-layer perceptron with only 2 layers containing 10 neurons in hidden layer. The network has 2 inputs, each for (Normalized-TF-IDF, History) and one output for pass or failing result. We used the logistic function as the activation function inside the neurons with backpropagation as the learning mechanism. Note that the NN model is designed to be simple and cheap, so that it is comparable with LR and

TABLE V
ACCURACY OF TWELVE FAULT PREDICTION TECHNIQUES IN THE DESKTOP, MOBILE AND TABLET FIREFOX – BL(BASELINE), LR(LINEAR REGRESSION), NLR(NONLINEAR REGRESSION), NN(NEURAL NETWORK), SH(SIMPLE HISTORY), AH(ALL HISTORY), WH(WEIGHTED HISTORY). FOR NN AVERAGE(AVG) AND STANDARD DEVIATION(SD) OF 10 RUNS ARE REPORTED.

| Type | Version | BL | | | LR | | | NLR | | | NN | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | SH | AH | WH | SH | AH | WH | SH | AH | WH | SH | | AH | | WH | |
| | | | | | | | | | | | AVG | SD | AVG | SD | AVG | SD |
| Desktop (TR) | 3.6 | 0.51 | 0.39 | 0.60 | 0.63 | 0.47 | 0.63 | 0.63 | 0.48 | 0.63 | 0.63 | 0.08 | 0.61 | 0.06 | 0.58 | 0.00 |
| | 4.0 | 0.45 | 0.54 | 0.50 | 0.69 | 0.67 | 0.66 | 0.69 | 0.67 | 0.67 | 0.68 | 0.00 | 0.64 | 0.01 | 0.66 | 0.00 |
| Desktop (RR) | 7.0 | 0.79 | 0.81 | 0.80 | 0.89 | 0.79 | 0.79 | 0.79 | 0.79 | 0.79 | 0.79 | 0.01 | 0.78 | 0.01 | 0.79 | 0.00 |
| | 8.0 | 0.87 | 0.86 | 0.86 | 0.88 | 0.87 | 0.88 | 0.88 | 0.87 | 0.88 | 0.88 | 0.01 | 0.86 | 0.00 | 0.87 | 0.00 |
| | 9.0 | 0.90 | 0.85 | 0.84 | 0.85 | 0.85 | 0.85 | 0.91 | 0.85 | 0.85 | 0.91 | 0.01 | 0.86 | 0.02 | 0.85 | 0.02 |
| | 10.0 | 0.85 | 0.86 | 0.85 | 0.87 | 0.85 | 0.85 | 0.86 | 0.85 | 0.85 | 0.86 | 0.01 | 0.85 | 0.00 | 0.85 | 0.00 |
| | 11.0 | 0.90 | 0.91 | 0.91 | 0.94 | 0.90 | 0.91 | 0.91 | 0.90 | 0.90 | 0.91 | 0.01 | 0.90 | 0.00 | 0.90 | 0.01 |
| | 12.0 | 0.91 | 0.91 | 0.91 | 0.89 | 0.91 | 0.91 | 0.91 | 0.91 | 0.91 | 0.91 | 0.01 | 0.91 | 0.00 | 0.91 | 0.00 |
| | 13.0 | 0.85 | 0.88 | 0.85 | 0.94 | 0.88 | 0.88 | 0.88 | 0.88 | 0.88 | 0.88 | 0.01 | 0.88 | 0.01 | 0.89 | 0.00 |
| Mobile | 18 | 0.83 | 0.74 | 0.83 | 0.80 | 0.83 | 0.83 | 0.80 | 0.82 | 0.82 | 0.81 | 0.01 | 0.81 | 0.02 | 0.83 | 0.02 |
| | 19 | 0.82 | 0.81 | 0.77 | 0.86 | 0.85 | 0.85 | 0.86 | 0.86 | 0.86 | 0.80 | 0.05 | 0.84 | 0.05 | 0.85 | 0.01 |
| | 20 | 0.85 | 0.76 | 0.83 | 0.82 | 0.83 | 0.84 | 0.83 | 0.82 | 0.84 | 0.83 | 0.01 | 0.84 | 0.00 | 0.85 | 0.00 |
| | 21 | 0.85 | 0.80 | 0.85 | 0.86 | 0.87 | 0.88 | 0.86 | 0.87 | 0.87 | 0.86 | 0.01 | 0.86 | 0.01 | 0.88 | 0.00 |
| | 22 | 0.87 | 0.75 | 0.83 | 0.86 | 0.86 | 0.86 | 0.86 | 0.86 | 0.88 | 0.87 | 0.01 | 0.87 | 0.00 | 0.87 | 0.00 |
| | 23 | 0.78 | 0.73 | 0.77 | 0.82 | 0.88 | 0.85 | 0.83 | 0.85 | 0.85 | 0.84 | 0.01 | 0.86 | 0.01 | 0.85 | 0.01 |
| | 24 | 0.80 | 0.69 | 0.76 | 0.88 | 0.88 | 0.88 | 0.88 | 0.88 | 0.88 | 0.87 | 0.02 | 0.88 | 0.00 | 0.87 | 0.00 |
| | 25 | 0.83 | 0.65 | 0.71 | 0.88 | 0.88 | 0.88 | 0.88 | 0.88 | 0.88 | 0.88 | 0.01 | 0.88 | 0.00 | 0.88 | 0.00 |
| | 26 | 0.82 | 0.61 | 0.68 | 0.94 | 0.94 | 0.94 | 0.94 | 0.94 | 0.94 | 0.93 | 0.03 | 0.94 | 0.01 | 0.93 | 0.01 |
| | 27 | 0.90 | 0.68 | 0.74 | 0.90 | 0.90 | 0.90 | 0.90 | 0.90 | 0.90 | 0.90 | 0.01 | 0.90 | 0.00 | 0.91 | 0.00 |
| | 28 | 0.91 | 0.63 | 0.77 | 0.96 | 0.97 | 0.97 | 0.96 | 0.97 | 0.97 | 0.93 | 0.01 | 0.96 | 0.00 | 0.96 | 0.00 |
| | 29 | 0.94 | 0.65 | 0.82 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.01 | 0.97 | 0.00 | 0.97 | 0.00 |
| Tablet | 18 | 0.88 | 0.83 | 0.88 | 0.88 | 0.89 | 0.89 | 0.88 | 0.89 | 0.89 | 0.88 | 0.01 | 0.87 | 0.01 | 0.88 | 0.01 |
| | 19 | 0.78 | 0.81 | 0.76 | 0.79 | 0.81 | 0.82 | 0.79 | 0.82 | 0.82 | 0.79 | 0.05 | 0.81 | 0.03 | 0.81 | 0.03 |
| | 20 | 0.88 | 0.77 | 0.80 | 0.89 | 0.89 | 0.89 | 0.89 | 0.89 | 0.89 | 0.88 | 0.01 | 0.86 | 0.01 | 0.85 | 0.01 |
| | 21 | 0.91 | 0.83 | 0.79 | 0.90 | 0.88 | 0.88 | 0.90 | 0.90 | 0.91 | 0.90 | 0.01 | 0.90 | 0.01 | 0.91 | 0.01 |
| | 22 | 0.85 | 0.75 | 0.85 | 0.85 | 0.84 | 0.84 | 0.85 | 0.84 | 0.85 | 0.85 | 0.01 | 0.84 | 0.02 | 0.85 | 0.02 |
| | 23 | 0.79 | 0.74 | 0.76 | 0.87 | 0.87 | 0.87 | 0.86 | 0.87 | 0.87 | 0.86 | 0.01 | 0.85 | 0.01 | 0.85 | 0.00 |
| | 24 | 0.81 | 0.66 | 0.71 | 0.94 | 0.94 | 0.94 | 0.93 | 0.94 | 0.94 | 0.93 | 0.02 | 0.92 | 0.01 | 0.92 | 0.01 |
| | 25 | 0.86 | 0.65 | 0.71 | 0.89 | 0.89 | 0.89 | 0.89 | 0.89 | 0.89 | 0.89 | 0.01 | 0.89 | 0.00 | 0.89 | 0.00 |
| | 26 | 0.83 | 0.61 | 0.67 | 0.94 | 0.94 | 0.94 | 0.94 | 0.94 | 0.94 | 0.94 | 0.02 | 0.94 | 0.01 | 0.94 | 0.00 |
| | 27 | 0.92 | 0.64 | 0.74 | 0.93 | 0.93 | 0.93 | 0.93 | 0.93 | 0.93 | 0.93 | 0.01 | 0.93 | 0.00 | 0.93 | 0.00 |
| | 28 | 0.93 | 0.62 | 0.81 | 0.95 | 0.95 | 0.95 | 0.95 | 0.95 | 0.95 | 0.95 | 0.01 | 0.95 | 0.00 | 0.95 | 0.00 |
| | 29 | 0.96 | 0.66 | 0.86 | 0.96 | 0.96 | 0.96 | 0.96 | 0.96 | 0.96 | 0.96 | 0.01 | 0.96 | 0.00 | 0.96 | 0.00 |

NLR. Therefore, tuning NN or using higher levels of hidden layers were not an objective of this study.

*7) Evaluation Metric:* In this study, accuracy of the predictions are used to evaluate the techniques (our proposed approaches and the baseline) over different releases. The accuracy metric is defined as 6:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

where,

**TP** True Positive
**TN** True Negative
**FP** False Positive
**FN** False Negative.

To compare the accuracy values of the baseline and the proposed approaches, we apply a non-parametric significant test, a matched pair Wilcoxons signed rank test [31], to determine if the difference between the accuracy results are statistically significant or not. After applying the test on the predicted output results, a p-value has been calculated over the releases of Desktop, Mobile, and Tablet projects, separately. The p-value threshold is set to 0.05. Note that in the case of NN, which is a randomized technique, we repeat each run 10 times and report the mean values. The significant test is then applied in the mean values over different releases, for each approach.

### B. Case study results

In this section, we explain and discuss the results of the experiment and answer our research questions.

*1) RQ1::* To answer this research question, we compare the simple history-based baseline method (SH-BL) with the Linear Regression as a standard statistical technique for classification, which uses both history-based metric and Normalized-TF-IDF. The idea is to see the effect of NLP without giving much advantages to the technique by using an advanced prediction model. Note that we can not use the baseline directly (i.e. witout a prediction method) with the NLP-based encoding, thus SH-LR, in this paper, is the simplest NLP-based prediction approach.

Looking at Fig. 2, our first observation is that overall the accuracies are high for both techniques. This shows that the history-based approach is quite effective in these releases (mostly rapid releases). However, the SH-LR shows even

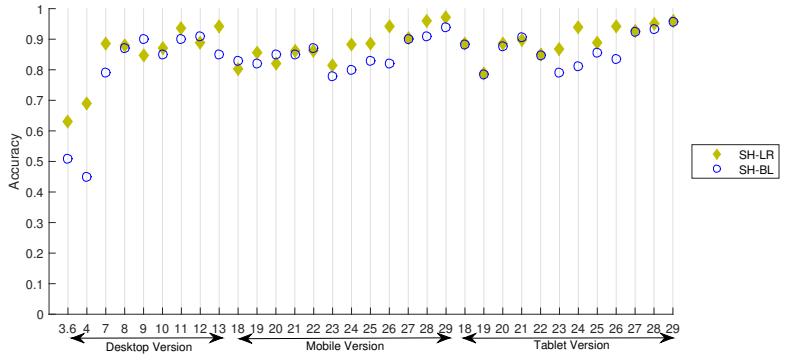


Fig. 2. Accuracy of Firefox Versions; Simple History Baseline (SH_Baseline) vs Simple History Linear Regression (SH_LR).

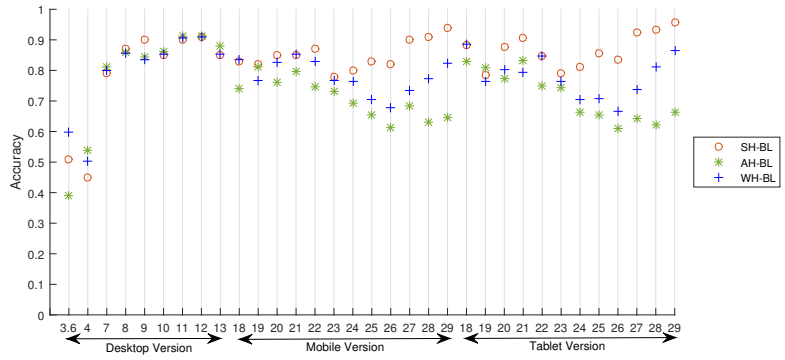


Fig. 3. Accuracy of Baseline (BL) based on History features; Simple History (SH), All History (AH) and Weighted History (WH).

higher accuracy. We can see accuracy improvements up to 24% (e.g., for version 4.0 Firefox Desktop). In addition, the low p-values reported in Table VI , 0.0004, show that the improvements are not by chance.

Another observation is that given that in many versions SH-BL already provides high accuracies, the average improvements provided by SH-LR are not that high (on average 3%, 3%, and 6% improvements, on Tablet, Mobile, and Desktop projects, respectively). However, the maximum improvements are more considerable: up to 12% (in version 24), 12% (in version 26), and 24% (in version 4) in Tablet, Mobile, and Desktop projects, respectively. It also worth mentioning that SH-BL outperforms SH-LR only in 6 cases out of 33 releases.

Therefore, the answer to our RQ1 is Yes! we can improve the basic history-based test case prediction using a simple NLP-base LR test case prediction model, even when it has already a high accuracy. This is quite interesting since it shows that the mined textual data brings extra knowledge that was not in the execution history. This knowledge is basically the coverage information which can not be seen in the history metric. However, the history measure that was used in RQ1 was quite naive and simple. So the next question will be whether a more complex history measure can help in prediction or not, which is studied in RQ2.

*2) RQ2::* The objective of RQ2 is to study the history-based measure in more details. In RQ2.1, we want to know whether SH-BL can be improved if we consider the entire history rather than only previous venison, or not. This basically tries to improve the baseline without using the NLP-based feature. In RQ2.2, we study the same question but in the context of our NLP-based LR method. In other words, we try to improve the SH-LR by a better history-based feature. To answer RQ2.1 and 2.2, we replace SH with AH and WH, as defined in Section III-B. Fig. 3 and 4 show the accuracy achieved by the baseline and LR models using the different historical attributes. Focusing on RQ2.1, (Fig. 3) shows that while all three modifications of history are almost the same in Desktop versions, Simple History works the best on Mobile and Tablet, meaning that the most significant information, in terms of simple baseline method, is in the last version and then in the more recent versions, explaining why Weighted History is outperforming All History. Note that the reason behind poor performance of SH in the first versions of Desktop is that those versions do not follow rapid release and each version lasts for almost a year. Therefore, the previous version's test execution results are outdated.

Looking at Fig. 4 to answer RQ2.2, we see that all three modifications of history-based features, in the context of

TABLE VI
P-VALUE OF BASELINE AND MODELING TECHNIQUES

| Method | | BL | | | LR | | | NLR | | | NN | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | SH | AH_ | WH | SH | AH | WH | SH | AH | WH | SH | AH | WH |
| BL | SH | 1.0000 | 0.0000 | 0.0003 | 0.0004 | 0.0025 | 0.0002 | 0.0001 | 0.0019 | 0.0000 | 0.0002 | 0.0049 | 0.0000 |
| | AH | 0.0000 | 1.0000 | 0.0004 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| | WH | 0.0003 | 0.0004 | 1.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| LR | SH | 0.0004 | 0.0000 | 0.0000 | 1.0000 | 0.2627 | 0.6389 | 0.4074 | 0.4781 | 0.5034 | 0.0811 | 0.0589 | 0.3498 |
| | AH | 0.0025 | 0.0000 | 0.0000 | 0.2627 | 1.0000 | 0.0627 | 0.5633 | 0.4080 | 0.0099 | 0.5698 | 0.0641 | 0.7935 |
| | WH | 0.0002 | 0.0000 | 0.0000 | 0.6389 | 0.0627 | 1.0000 | 0.4237 | 0.3061 | 0.0424 | 0.0656 | 0.0028 | 0.4106 |
| NLR | SH | 0.0001 | 0.0000 | 0.0000 | 0.4074 | 0.5633 | 0.4237 | 1.0000 | 0.8484 | 0.0363 | 0.0687 | 0.0261 | 0.5372 |
| | AH | 0.0019 | 0.0000 | 0.0000 | 0.4781 | 0.4080 | 0.3061 | 0.8484 | 1.0000 | 0.0010 | 0.5178 | 0.0548 | 0.8370 |
| | WH | 0.0000 | 0.0000 | 0.0000 | 0.5034 | 0.0099 | 0.0424 | 0.0363 | 0.0010 | 1.0000 | 0.0032 | 0.0001 | 0.0589 |
| NN | SH | 0.0002 | 0.0000 | 0.0000 | 0.0811 | 0.5698 | 0.0656 | 0.0687 | 0.5178 | 0.0032 | 1.0000 | 0.3595 | 0.4774 |
| | AH | 0.0049 | 0.0000 | 0.0000 | 0.0589 | 0.0641 | 0.0028 | 0.0261 | 0.0548 | 0.0001 | 0.3595 | 1.0000 | 0.0342 |
| | WH | 0.0000 | 0.0000 | 0.0000 | 0.3498 | 0.7935 | 0.4106 | 0.5372 | 0.8370 | 0.0589 | 0.4774 | 0.0342 | 1.0000 |

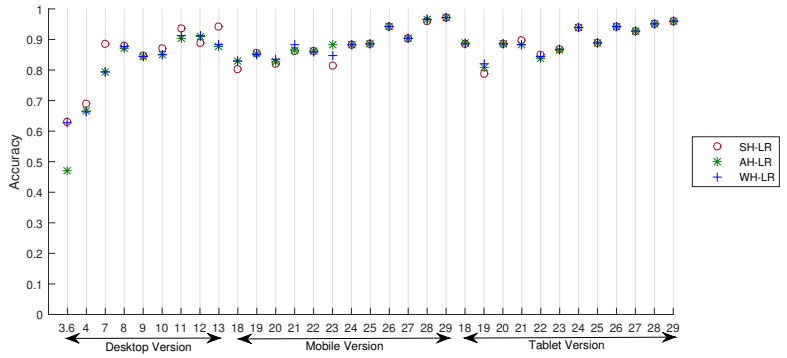


Fig. 4. Accuracy of Linear Regression (LR) based on History features; Simple History (SH), All History (AH) and Weighted History (WH).

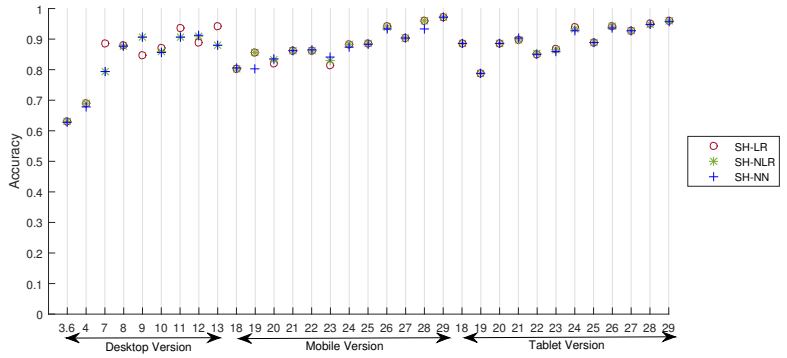


Fig. 5. Accuracy of different learning methods; SH-LR, SH-NLR and SH-NN.

the NLP-based LR model, have the same level of accuracy, indicating that expanding history from one version to all previous versions, does not influence the learning accuracy of the NLP-based model. The above conclusion can also be drawn by looking at high p-values between SH-LR with WH-LR and AH-LR which are 0.63 and 0.26.

Therefore, given that the cost of collecting AH and WH features are higher than SH and they do not provide better accuracies, in most cases, our answer to RQ2 is: NO! Looking at the richer history does not improve the prediction accuracies, at least when using BL or LR, as the prediction model. Now the follow up question is whether improving the prediction model helps in improving the accuracy or not, which is studied in RQ3.

*3) RQ3::* In this RQ, we focus on the learning side of the approach. So we keep the SH-LR as our default history measure and replace LR with NLR and NN models.

Fig. 5 compares these three techniques, in terms of accuracy. As a general observation, we can see that replacing the learning algorithm does not provide any significant changes

compared to SH-LR, which means that the output of the model, i.e., the failure of the test cases, can be learned without a complex model as well. Table V shows the results for all 12 combinations of the three history-based features and four prediction methods. The conclusions drawn from those parts of the table that are related to Fig. 5 is that replacing LR with an NLR or NN-based approaches does not help improving the accuracies of the predictions. This is also clear from the p-value results in Table VI, where comparing SH-LR with none of the *-NLR and *-NN approaches result in lower than threshold p-values.

Given that the more complex prediction models also come with more cost and tuning needs, our conclusion is that extracting keywords of the features under test from the textual data in test cases is possible through a simple NLP-based approach (POS+TF-IDF). This representation combined with a basic history-based measure (historical test execution results from previous release) can potentially provide a highly accurate test case failure prediction approach, using a linear regression classifier.

## V. Conclusion and Future Work

Manual testing is an expensive and tedious software testing activity, which is essential for getting users perspective on the released features. In this paper, we proposed an NLP-based approach (POS) for extracting nouns from textual data of test cases (manual test instructions) to represent each test case. We then proposed an IR-based measure (TF-IDF) to be assigned to each test case. Finally, we used three different prediction models to predict failure of test cases using this measure and three different variations of history-based measures, separately. Our results based on an empirical study of 41 releases of Mozilla Firefox's manual test suites showed that a simple history-based feature (only previous release data) combined with POS/TF-IDF data in a linear regression model can accurately predict test cases' failure in new releases. In addition, the NLP-based approach can provide even more improvement on the accuracy of predictions by the baseline approach (only using history). To the best of our knowledge, this work is the first use of NLP on manual test case scripts for test failure prediction and has shown promising results, which we are planning to replicate on different systems and expand on different NLP-based features to more accurately extract features keywords from test cases.

## Acknowledgment

Part of this work was done when the authors were employed by the University of Manitoba. This work is partially supported by the Natural Sciences and Engineering Research Council of Canada.

## References

[1] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: an empirical study," in *Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference on*, 1999, pp. 179–188.

[2] A. Hudaib, B. Hammo, and Y. Alkhader, "Towards software requirements extraction using natural language approach," in *Proceedings of the 6th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems*, ser. SEPADS'07. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2007, pp. 155–160. [Online]. Available: http://dl.acm.org/citation.cfm?id=1353801.1353829

[3] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 675–689, Sep 1999.

[4] N. Nagappan, L. Williams, M. Vouk, and J. Osborne, "Early estimation of software quality using in-process testing metrics: a controlled case study," in *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4. ACM, 2005, pp. 1–7.

[5] S. Elbaum, G. Rothermel, and J. Penix, "Techniques for improving regression testing in continuous integration development environments," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 235–245.

[6] T. B. Noor and H. Hemmati, "A similarity-based approach for test case prioritization using historical failure data," in *Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on*. IEEE, 2015, pp. 58–68.

[7] H. Hemmati, Z. Fang, and M. V. Mantyla, "Prioritizing manual test cases in traditional and rapid release environments," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, April 2015, pp. 1–10.

[8] H. Hemmati, Z. Fang, M. V. Mntyl, and B. Adams, "Prioritizing manual test cases in rapid release environments," *Software Testing, Verification and Reliability*, vol. 27, no. 6, pp. e1609–n/a, 2017, e1609 stvr.1609. [Online]. Available: http://dx.doi.org/10.1002/stvr.1609

[9] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, ser. NAACL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 173–180. [Online]. Available: https://doi.org/10.3115/1073445.1073478

[10] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, Nov. 1975. [Online]. Available: http://doi.acm.org/10.1145/361219.361220

[11] K. Koga, "Software reliability design method in hitachi," in *Proc. Third European Conference on Software Quality*, 1992.

[12] T. B. Noor and H. Hemmati, "Test case analytics: Mining test case traces to improve risk-driven testing," in *Software Analytics (SWAN), 2015 IEEE 1st International Workshop on*. IEEE, 2015, pp. 13–16.

[13] D. Mondal, H. Hemmati, and S. Durocher, "Exploring test suite diversification and code coverage in multi-objective test case selection," in *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 1–10.

[14] M. Harman, "Making the case for morto: Multi objective regression test optimization," in *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*. IEEE, 2011, pp. 111–114.

[15] S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein, "Static test case prioritization using topic models," *Empirical Software Engineering*, vol. 19, no. 1, pp. 182–212, Feb 2014. [Online]. Available: https://doi.org/10.1007/s10664-012-9219-7

[16] R. C. Bryce and A. M. Memon, "Test suite prioritization by interaction coverage," in *Workshop on Domain specific approaches to software test automation: in conjunction with the 6th ESEC/FSE joint meeting*. ACM, 2007, pp. 1–7.

[17] R. C. Bryce and C. J. Colbourn, "Prioritized interaction testing for pairwise coverage with seeding and constraints," *Information and Software Technology*, vol. 48, no. 10, pp. 960–970, 2006.

[18] D. R. Kuhn, R. N. Kacker, and Y. Lei, *Introduction to combinatorial testing*. CRC press, 2013.

[19] S. Sampath, R. C. Bryce, G. Viswanath, V. Kandimalla, and A. G. Koru, "Prioritizing user-session-based test cases for web applications testing," in *Software Testing, Verification, and Validation, 2008 1st International Conference on*. IEEE, 2008, pp. 141–150.

[20] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Not my bug! and other reasons for software bug report reassignments," in *Proceedings of the ACM 2011 conference on Computer supported cooperative work*. ACM, 2011, pp. 395–404.

[21] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang, "Automated support for classifying software failure reports," in *Software Engineering, 2003. Proceedings. 25th International Conference on*. IEEE, 2003, pp. 465–475.

[22] G. Denaro and M. Pezzè, "An empirical evaluation of fault-proneness models," in *Proceedings of the 24th International Conference on Software Engineering*. ACM, 2002, pp. 241–251.

[23] S. Samarasinghe, *Neural networks for applied sciences and engineering: from fundamentals to complex pattern recognition*. CRC Press, 2016.

[24] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: an introduction," *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 544–551, 2011. [Online]. Available: + http://dx.doi.org/10.1136/amiajnl-2011-000464

[25] D. Das and S. Petrov, "Unsupervised part-of-speech tagging with bilingual graph-based projections," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 2011, pp. 600–609.

[26] A. Søgaard, "Simple semi-supervised training of part-of-speech taggers," in *Proceedings of the ACL 2010 Conference Short Papers*, ser. ACLShort '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 205–208. [Online]. Available: http://dl.acm.org/citation.cfm?id=1858842.1858880

[27] S. L. Abebe and P. Tonella, "Natural language parsing of program element names for concept extraction," in *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*. IEEE, 2010, pp. 156–159.

[28] M. Saeki, H. Horai, and H. Enomoto, "Software development process from natural language specification," in *Software Engineering, 1989. 11th International Conference on*. IEEE, 1989, pp. 64–73.

[29] A. Marcus, "Semantic driven program analysis," in *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*. IEEE, 2004, pp. 469–473.

[30] Y. Feng, J. A. Jones, Z. Chen, and C. Fang, "Multi-objective test report prioritization using image understanding," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2016, pp. 202–213.

[31] R. L. McCornack, "Extended tables of the wilcoxon matched pair signed rank statistic," *Journal of the American Statistical Association*, vol. 60, no. 311, pp. 864–871, 1965.