Measuring Requirement Quality to Predict Testability

Jane Huffman Hayes   Wenbin Li   Tingting Yu   Xue Han   Mark Hays   Clinton Woodson
Department of Computer Science
University of Kentucky USA
hayes@cs.uky.edu, wenbin.li@uky.edu, tyu@cs.uky.edu, xue.han@uky.edu, mark.hays@rose.hulman.edu, clint.woodson@uky.edu

***Abstract*—Software bugs contribute to the cost of ownership for consumers in a software-driven society and can potentially lead to devastating failures. Software testing, including functional testing and structural testing, remains a common method for uncovering faults and assessing dependability of software systems. To enhance testing effectiveness, the developed artifacts (requirements, code) must be designed to be testable. Prior work has developed many approaches to address the testability of code when applied to structural testing, but to date no work has considered approaches for assessing and predicting testability of requirements to aid functional testing. In this work, we address requirement testability from the perspective of requirement understandability and quality using a machine learning and statistical analysis approach. We first use requirement measures to empirically investigate the relevant relationship between each measure and requirement testability. We then assess relevant requirement measures for predicting requirement testability. We examined two datasets, each consisting of requirement and code artifacts. We found that several measures assist in delineating between the testable and non-testable requirements, and found anecdotal evidence that a learned model of testability can be used to guide evaluation of requirements for other (non-trained) systems.**

***Index Terms*—requirement testability, code testability, machine learning, supervised classification learning, statistical analysis, correlation analysis, subjective assessment, human analyst**

I. INTRODUCTION

A nascent aviation industry would never have dreamed of civilians riding in commercial planes just a short time after the first successful manned flight, yet the software industry seems to have no difficulty releasing software of questionable quality levels. The comparison may seem extreme – crashed airplanes cost lives. But in 2015, failed computer programs can also cost lives, cost large amounts of money, risk the privacy or identity of consumers, and/or cause severe damage to the environment. Thus, the quality and reliability of systems must be assured before the system is deployed to the field. Testing (including functional and structural testing) is still the most commonly used approach to assess the quality of software systems.

Software testability has been an important issue for software testing and verification [1]. Software is built upon product requirements, and requirements information has been used in functional testing to ensure that a program meets its requirements. Therefore, it is important to develop testable requirements to improve the effectiveness of functional testing. However, most work to date has primarily focused on code-level testability, including both static [2] and dynamic measures [1]. While the importance of using requirements to aid functional testing has been recognized by the requirements engineering community [3][4], little work has addressed the problem of assessing and predicting testability of requirements.

Binder [5] provided a number of definitions of requirement testability from various sources " A requirement is considered to be testable if an objective and feasible test can be designed to determine whether the requirement is met by the software. [6]" Also, defined in IEEE standard 610.12, requirement testability is "The degree to which a requirement is stated in terms that permit establishment of test criteria and performance of tests to determine whether those criteria have been met [7]." If requirements are written to support testing, the code that is developed to implement the requirements should be easier to test. This should in turn lead to higher quality software.

Analogous to the use of code metrics (e.g., assertions, cyclomatic complexity) to predict code-level testability, we posit that existing requirements quality measures such as understandability/readability can be useful in predicting requirement testability. Enlightened developers ensure that requirements, generally written in natural language text, are consistent and complete and readable; often using automated tools. Automated tools can also be used to easily and quickly capture requirement quality measures.

In this paper, we present an approach for assisting software engineers to assess the testability of requirements using a learned model combining a number of static quality measures found to have some level of correlation with testability. Specifically, we investigate the relationship between each static measure (X) and testability, where testability is manually determined based on X's definition. We then identify measures that affect requirement testability, and use logistic regression analysis to evaluate the capability of these measures to predict requirement testability. We use the measures to learn a model of testability, assessing the accuracy of the model on the two datasets (used to train and test) as well as anecdotally on a third non-trained dataset. Our hope is that we can ultimately help requirements and software engineers to determine if the requirements have been specified in a testable way before proceeding to later phases of the development lifecycle. In our empirical study we use two real world subjects.

The paper is organized as follows. Section II discusses requirement testability. Section III presents the study design, correlation analysis, logistic regression, supervised learning, and threats to validity. Section IV discusses results and analysis. Related work is presented in Section V. Section VI discusses conclusions and future work.

II. Testability

This section presents definitions and measures of requirement testability.

A. Definitions of Requirement Testability

While a number of definitions of requirement testability have been provided by various sources, they are all concerned with the degree to which tests can be created, executed, and verified against expected output. We therefore posit that a requirement is testable if a complete testing scenario used in functional testing can be found. A testing scenario includes: 1) test input, 2) test execution, and 3) expected output. A test input is a condition used to direct how a system under test behaves. A test execution is a sequence of steps to execute a test input. An expected output is the correct system behavior under a test input.

Consider an example requirement "*If a user clicks the "send" button, an email must be sent within 5 seconds.*" This requirement is testable because all three aforementioned conditions are met. Specifically, "send" is an input, a test execution involves sending an email, and the expected output is "email sent within 5 seconds." Since measuring requirement testability is a subjective and difficult task, this process often involves a human in the loop.

B. Measurement of Requirement Testability Factors

We hypothesize that several factors can potentially influence requirement testability. These factors in particular appear to be: 1) the degree to which a requirement is readable, 2) the number of predicates in a requirement, and 3) the number of total/complex words used in a requirement (described in Section III). By leveraging these factors, we should be better able to create effective models to predict requirement testability.

We collected several measures to assess readability of the requirements:

Average Grade Level (*agl*)
Flesch Kincaid Reading Ease (*fkre*)
Flesch Kincaid Grade Level (*fkgl*)
Gunning Fog Score (*gfs*)
SMOG Index (*smog*)
Coleman Liau Index (*cli*)
Automated Readability Index (*ari*)

Low readability of a requirement implies that it may not be testable. A subset of these measures, and the tool we used for collection, are further explained in Section III. B.

The next measure we used in this study is *number of predicates*. Predicates specify what subjects are or what they do. For example, given a requirement "*the user can edit the Style Preferences through the Style Preference GUI interface*," the verb "*edit*" is a predicate that describes what the user can do. If a requirement does not contain any predicates (or few), then it may not be testable.

III. Study

This section presents the study design, techniques, and threats to validity.

To assess our approach we consider two research questions.

RQ1 - Do quality measures that characterize the understandability and quality of requirements also characterize testable requirements?

RQ2 – Can a model of requirement testability be learned and applied to other requirements?

The first research question explores the relationship between quality measures and testability and determines the predictive ability of each quality measure. The second research question lets us investigate whether existing requirement quality measures can be modeled and assist in categorizing requirement testability.

A. Objective of Analysis

As objects of analysis we chose two real world data sets - Browser and iTrust [8]. The two datasets have been widely used in other requirements research. We make use of their requirements, code, and trace links between their requirements and code. Browser[1] is a web browser written in Java by the École Polytechnique de Montréal. It consists of 24 requirements and 52 code modules. iTrust is a medical application that maintains patient medical history and records and permits communication with Doctors. It consists of 59 use cases and 11 code modules. It is written in Java.

To address our research questions, we required labels for the requirements as testable or not. We asked three of the co-authors (in isolation) to assign scores from 1 to 5 to each requirement, where 5 is easy to test and 1 is hard to test. The remaining co-authors examined the labels and assessed inter-rater agreement. For both datasets, the agreement was very low.

The remaining co-authors thus looked at a number of voting strategies. Using majority rule, it was found that approximately 66% of the labels could be assigned. For requirements with no majority rule, the labels of the most experienced co-author were selected (has worked in requirements area and specifically with natural language processing and consistency checking of requirements for eight years). A binary label was also assigned; labels of 1, 2, 3 were given a 0 for testability and 4, 5 labels were tagged as 1. The original labels (1 – 5) were then re-examined and it was found that roughly 10% of the labels were changed (0 to 1 or vice versa) due to use of the experienced co-author labels.

In addition, the remaining co-authors examined the agreement on the "best" and "worst" requirements (those labeled 5 and 1, see Fig. 1 in Appendix for examples). It was

[1] We studied 10 of the 24, for which trace links to code were provided by the authors

found that the three co-authors had strong agreement on these labels, thus some analysis used these entries. Models were built and analysis was undertaken (using the spreadsheets and arff files), as described below.

Table 1 summarizes the distribution of testable (labeled as 1) and non-testable requirements (labeled as 0). As the table shows, for iTrust and Browser, over 70% of the requirements were categorized as testable. Tables 2 and 3 (in appendix) provide descriptive statistics for a subset of the measures for iTrust and Browser.

TABLE 1. DISTRIBUTION OF TESTABLE REQUIREMENTS

Systems	Testable Reqts	%
iTrust	18	75
Browser	8	72.73

B. *Quality Measures*

To answer our research questions, we also required requirement quality measures [9]. Flesch-Kincaid reading ease and grade level (*fkre* and *fkgl*) both examine the number of words and syllables in a requirement. Flesch-Kincaid reading ease estimates how easy it will be to understand the requirement. Flesch-Kincaid grade level estimates the grade level at which the text has been written (i.e., a result of 6 indicates that the text was written so that a typical 6th grader can understand it). Coleman-Liau index (*cli*) uses the number of characters and words in a requirement to estimate the grade level of the text. Gunning Fog score (*gfs*) is calculated by looking at the ratio of words to complex words. A complex word is any word with three or more syllables. SMOG index (*smi*) uses a more accurate syllable count. Automated readability index (*ari*) is similar to *cli* but uses a different ratio between number of characters and words. All of the aforementioned measures are used often in the education field. Average grade level (*agl*) is calculated as the average of *fkgl*, *cli*, *smog*, *ari*, and *gfs*. The number of predicates (*pred*), the number of words in each requirement (*now*), and number of complex words (*nocw*) were also measured.

We believe that some measures are redundant with others and thus can be eliminated. These measures do not provide insights into our research questions. To eliminate redundant measures, we identified several groups of measures such that any two measures in the same group are strongly correlated – correlation analysis was performed as described in Section C. We believe that each group can be represented by one or two measures within the group.

Specifically, we identified three groups of measures. The first group includes five measures: *fkgl*, *gl*, *fkre*, *gfs*, and *ari*. We selected *fkgl* and *fkre* to represent these five measures. The second group contains two measures: *cli* and *smi,* we selected *cli*. The third group contains three measures: *now*, *nocw*, and *nos*, and we used *now* and *nocw* to represent this group. The measure *pred* is not correlated with any measures and thus was selected. This process yielded us six measures: *fkgl, fkre, cli, now, nocw, and pred*.

C. *Data Analysis Techniques*

We chose machine learning and statistical analysis techniques to allow us to answer each of our research questions.

1) Correlation Analysis. To address our research question #1, we first apply correlation analysis to help distinguish what aspects of a requirement played a role in its testability. Correlation analysis estimates a sample correlation coefficient, which ranges from -1 to 1 and quantifies the direction and strength of the linear association between a dependent variable (testability) and an independent variable (quality measures). A correlation coefficient of 1 represents a perfect positive linear relationship. When using correlation analysis in our study, the values of dependent variables are testability labels (i.e., 0, 1). In this study we saw very low correlation and thus interpret correlation coefficients less than -0.3 as a sign of weak negative linear relationship.

2) *Logistic Regression Analysis.* To further explore RQ1, we apply logistic regression (LR) to examine the ability of each quality measure to predict a binary response for requirement testability. LR is a standard statistical model used to measure the relationship between *categorized* dependent variables and one or more independent variables. Univariate LR involves one independent variable; multivariate LR involves one or more independent variables. In our study, the dependent variable is requirement testability with binary labels. We used both univariate and multivariate LRs to find the relationship between static measures and requirement testability. The univariate analysis allows us to find individual effect of each static measure on testability, whereas multivariate analysis is used to evaluate the combined effects of all static measures on testability.

To evaluate the performance of our predicated model, we used ROC (receiver operating characteristic) curve, where area of 1 represents a perfect model, and that of 0.5 or below represents a random model.

3) *Supervised Learning – Decision Tree.* To answer our second research question, we performed N-fold cross validation (where N was 10 whenever possible) using the Weka J48 classifier to analyze the data. J48 is an implementation of the C4.5 decision tree algorithm [10]. A decision tree can be used to decide the value of a dependent variable (in our case, testable) based on a set of independent variables. The root of a decision tree represents that the C4.5 algorithm begins with the original dataset. Each internal node in a decision tree indicates an iteration in which C4.5 chooses the unused attribute that most effectively splits the data. The leaves show the values of the dependent variable.

We determine the accuracy of the classification based on the following results given by Weka: confusion matrix, detailed accuracy by class, correctly classified instances, incorrectly classified instances, and Kappa statistic.

The confusion matrix, detailed accuracies, and the correctly/incorrectly classified instances show the raw numbers and the percentage of the instances that are correctly/incorrectly classified. In our case we have 2*2 confusion matrixes because we have two classes: testable and non-testable. For example, a confusion matrix

```
a      b
5      15 |   a = testable
10     20 |   b = non-testable
```

shows the raw numbers of the instances that are true positive (5), false positive (10), false negative (15), and true negative (20). The matrix means that five testable instances and 10 non-testable instances are classified as testable, while 15 testable instances and 20 non-testable instances are classified as non-testable. In total, 25 (50%) instances are correctly classified, and the other 25 (50%) instances are incorrectly classified.

The detailed accuracies are calculated based on the confusion matrix for each class. For the class testable, the True Positive (TP) rate measures the proportion of testable instances which were correctly classified as testable: 5/(5+15) = 0.25. The False Positive (FP) rate measures the proportion of non-testable instances which were incorrectly classified as testable: 10/(10+20) = 0.33. The Precision measures how many testable instances determined by the classifier are truly testable: 5/(5+10) = 0.33. The Recall measures how many testable instances are correctly identified. The Recall is equivalent to the TP rate. The F Measure is the harmonic mean of Precision and Recall: 2*(0.33*0.25)/(0.33+0.25) = 0.28. A ROC curve is constructed by using the TP rate and FP rate.

The Kappa statistic [14] measures the agreement between the classifications with the true classes. It takes the agreement that may be due to chance into account.

D. *Study Operation*

We used Read-able.com [11] to collect measures for the requirements, including *agl, fkre, fkg, gfs, smog, cli*, and *ari*. We used a natural language processing technique Semantic Role Labeling (SRL) [12] to detect the predicates in natural language requirements. SRL identifies predicates together with the semantic arguments that are related to these predicates, and classifies the arguments into different roles. In the requirement "*the user can edit the Style Preferences through the Style Preference GUI interface*," the role of "*user*" is subject, the role of "*Style Preferences*" is object, and the role of "*Style Preference GUI interface*" is the manner in which the "*edit*" action is performed. In this study we used the SRL tool Senna [13] to process natural language requirements and calculate the number of the identified predicates. All measures were collected into spreadsheets for analysis and were used to build arff files for Weka [14]. Our correlation analysis and LR analysis were performed by using Microsoft Excel and its XLSTAT plugin. The supervised learning analysis was performed on Weka.

E. *Threats to Validity*

The primary threat to external validity for this study involves the representativeness of datasets (i.e., software systems, requirements). Other datasets may be larger than those studied and may exhibit different behaviors and complexity that can affect requirement quality. However, the datasets we utilized are popular and real; the systems we studied are real-world applications with requirements available. Also, we studied systems from two different domains.

The primary threats to internal validity for this study are possible mistakes in the study operations and possible faults in the tools that we used to perform evaluation. We controlled for these threats by first extensively validating our results for a smaller set of requirements for which we could determine correct results. We also chose to use popular and established tools (e.g., Weka, Senna, XLSTAT) for gathering data and performing analysis. A second major source of potential bias involves manually determining requirement testability. To reduce this threat, we had numerous researchers evaluate each requirement, in isolation, and then applied various techniques to increase confidence in the requirement testability labels.

Where construct validity is concerned, our measurements of requirement quality focus on readability/understandability. However, there are other metrics that could be considered, such as completeness, size, volatility, and traceability. The costs of applying the approach and the human costs of requirements inspection are also of interest. We reduced such threats by applying statistical analysis, ensuring that we checked the assumptions of each test we applied. In addition, we saw disagreement among our expert raters. Their testability ratings are inherently subjective and might not be reproducible. As mentioned, in cases of disagreement, we deferred to our most experienced rater to mitigate this threat.

IV. Results and Analysis

This section discusses the results of correlation analysis, logistic regression, and supervised learning.

A. *Correlation Analysis*

For the iTrust dataset, the correlation coefficient between the measure *now* and testability has the highest absolute value: 0.33. Because the correlation coefficient is -0.33, we believe that there is a weak negative linear relationship between *now* and testability. Other measures that have high absolute value correlation coefficients include *fkre* (-0.18) and *fkgl* (0.15), but their absolute values are not high enough to be a sign of correlation.

For the Browser dataset, the correlation coefficient between *now* and testability is only -0.11, which is not a strong sign of correlation. However, the correlation coefficient between *nocw* and testability is -0.38. This means that *nocw* and testability may have a weak negative linear relationship. The next largest absolute correlation coefficients are for *fkre* (0.23) and *fkgl* (-0.22). However, as with the iTrust dataset, the evidence of correlation between these two measures and testability is not strong. The relationships between *now/nocw* and testability are reasonable. Requirements with more words and complex words are likely to be less readable, and requirements with low readability may be hard to understand and test.

B. Logistic Regression

In a regression model, each label b_i is an estimated coefficient corresponding to independent variables. The larger the absolute value of b_i, the stronger the effect of the static measure on predicting high requirement testability. The p-value (-2Log Likelihood) is used to assess the significance of coefficients; we chose significance level 0.05. A measure is highly predictive if its coefficient is significantly different from zero (b>0.1) in terms of the p-value.

TABLE 4. UNIVARIATE LR ANALYSIS

Systems	iTrust			Browser		
Measures	b_i	p	AUC	b_i	p	AUC
fkre	**0.663**	**0.037**	**0.824**	0.340	0.346	0.708
fkgl	0.438	0.094	0.806	0.568	0.163	0.792
cli	**1.348**	**0.003**	**0.889**	0.024	0.949	0.542
now	0.208	0.391	0.583	0.062	0.028	0.5
nocw	**0.168**	**0.05**	**0.62**	0.062	0.868	0.5
pred	0.029	0.911	0.509	**0.369**	**0.035**	**0.667**

Table 4 shows the results of univariate LR analysis for each measure (reduced to eliminate collinearity) across the two datasets. For four out of six measures, their coefficients are significant (numbers indicated by bold font) for at least one dataset. The effectiveness scores of their predicted models were also high (AUC > 0.5). Only *fkgl* and *now* did not demonstrate significance on either dataset. In fact, each of the four measures performed well on only one dataset. Our results imply that all five measures except for *fkgl* and *now* are related to testability. The extent to which a quality measure can be used to predict testability also varies across applications.

TABLE 5. MULTIVARIATE LR ANALYSIS

	b_{kfre}	b_{kfgl}	b_{cli}	b_{now}	b_{nocw}	b_{pred}	p	AUC
iTrust	2.25	1.00	*2.31*	0.45	2.19	0.05	0.04	0.94
Broswer	1.95	0.06	*1.98*	0.53	0.22	0.56	0.05	0.88

We next present the results obtained using multivariate LR analysis to illustrate how well requirement testability can be predicted when the static measures are used in combination – see Table 5. As the table shows, for both datasets, the coefficient values are significant. Figure 2 and Figure 3 display the ROC curves for the combined measures for the two datasets. On both datasets, the AUC values are higher than those reported in univariate LR analysis. This indicates that the predicted model with combined measures is more effective than that with single measures. We next inspect the degree to which each measure can influence the requirement testability. Among the six measures, on each dataset, *cli* reported the highest coefficient score (numbers indicated by italic font). On the other hand, *pred* and *fkgl* reported the lowest scores for iTrust and Browser, respectively. While additional study is needed, if the results can generalize, *cli* is the most influential measure to requirement testability.

Based on our findings, we believe the response to RQ1 is "Yes, static measures that characterize the understandability and quality of requirements also characterize testable requirements," but more study is needed.

C. Supervised Learning

We applied supervised learning to address RQ2. A model will assist in categorizing newly written or modified requirements as testable, easily validated by functional testing, or not. Using measures we believe to be correlated with requirement testability labels, we used Weka to perform classification. Specifically, we used the J48 learner (implementation of C4.5, discussed in Section III). The intent is to apply the learned model to not yet evaluated requirements. Basically, a practitioner would need to only collect static measures for the requirements and then plug the static measures into the model (or use Weka to do so) to determine if the requirement appears to be testable. If not, information from the model (such as the thresholds for the leaves of the classification tree) can be used to help improve the text of the requirement.

In planning the study, we anticipated an unbalanced data challenge. We hypothesized that most requirements would be non-testable. However, the Browser and iTrust requirements were almost all testable. To address our minority class (non-testable requirements), we undertook a number of steps: boosting, bagging, and blending (called stacking in Weka). The learned model still had very low accuracy for the minority class (we could achieve high accuracy for the majority class, but the model always labeled the non-testable instances as testable).

Also, we found that we had too many measures in some cases with respect to the number of data records we had, especially for Browser. We thus decreased the measures by only using one or two of the measures to represent a set of measures that correlated with each other (detected using correlation analysis described in Section III.B.). The learned model was able to achieve 69% accuracy in predicting the testability label. The pruned tree that resulted is shown below (denoted Model 1):

```
pred-adj <= 7
|  fkgl <= 13
|  |  nocw <= 1: testable (4.0)
|  |  nocw > 1
|  |  |  fkre <= 76
|  |  |  |  nocw <= 4: testable (4.0)
|  |  |  |  nocw > 4
|  |  |  |  |  fkre <= 55.1: testable (2.0)
|  |  |  |  |  fkre > 55.1: not-testable (2.0)
|  |  |  fkre > 76: not-testable (2.0)
|  fkgl > 13: not-testable (3.0)
pred-adj > 7: testable (6.0)
```

It is interesting that *cli* was not in the model. When we examined all requirements of iTrust and Browser (as agreed by the researchers), this model resulted (denoted Model 2),

```
cli <= 12.4
|  fkgl <= 11.3
|  |  fkre <= 52.6: 1 (27.0)
|  |  fkre > 52.6: 0 (35.0)
|  fkgl > 11.3: 1 (58.0)
cli > 12.4: 0 (26.0)
```

with the following accuracy (described in Section III.C):

```
Correctly Classified Instances       143         97.9452 %
Incorrectly Classified Instances       3          2.0548 %
Kappa statistic                    0.9575
Mean absolute error                  0.0224
Root mean squared error               0.1411
Relative absolute error              4.5983 %
Root relative squared error          28.5944 %
Total Number of Instances             146

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Prec   Recall  F-Meas  ROC Area  Class
            0.951    0      1      0.951   0.975    0.994     0
            1       0.049   0.966   1      0.983    0.994     1
 Wt.Avg   0.979    0.029   0.98    0.979   0.979    0.994
```

It is notable that *cli* was a part of the learned model.

To determine if a model learned on iTrust and Browser could be used to predict testable requirements for a different project, we examined a student project called CardPlay-D. This project is an Android application used for playing cards; it consists of 43 requirements and 39 code modules. CardPlay-D had 22 testable requirements and 21 non-testable requirements, a much different distribution than Browser and iTrust. Applying Model 1, we looked at each individual measure in the model as well as the overall model (decision tree). We found that all CardPlay-D requirements have *pred* less than or equal to 7 (recall that >=7 means testable). All requirements except one have *fkgl* less than or equal to 13 (where <= 13 means testable). Fourteen were incorrectly predicted, but 26 were correctly predicted as testable (65% correct). For *fkre*, 19 requirements were predicted correctly as testable (less than or equal to 55.1) but five were not – or 73.6%. When the rules are used together (without *nocw*, which we did not have for the student project), nine requirements are predicted to be testable. Six were predicted correctly (or 66.67% correct).

Looking at Model 2, for *cli* less than or equal to 12.4, there are nine requirements predicted incorrectly and 17 correctly (or 65%). For *fkgl* <=11.3, there are 13 requirements predicted incorrectly and 26 predicted correctly or 66%. For *fkre* <= 52.6, three are predicted incorrectly and seven correctly (70%). When all three rules are applied (*cli*, *fkgl*, *fkre*,) no requirements are selected.

Our preliminary results show that there is promise of being able to answer RQ2 "Yes, we can learn a model of requirement testability." However, our results are not conclusive and further analysis with more datasets is required.

V. Related Work

We address requirement testability, code testability, and earlier application of machine learning to requirements engineering problems.

Boehm [16] discusses the reasons for software requirements and design specifications to be verified and validated. He defines the terms "verification" and "validation," as well as explains their context in the software life-cycle. Also he evaluates the relative cost and effectiveness of a software requirement. In talking about the basis of a good requirement, he gives the four basic criteria of a requirement: completeness, consistency, feasibility, and testability.

Along the lines of requirement criteria, Davis et al. [15] elaborate the important characteristics of a quality Software Requirements Specification, including but not limited to: unambiguous, complete, verifiable, precise, and traceable. Measures are proposed for the attributes. It is noted that measuring understandability is difficult and a subjective measure is proposed based on readers' belief related to understanding the requirement.

Binder [5] speaks about testability in object-oriented development. He suggests that one "map the testability terrain for object-oriented development to assist the reader in finding relatively shorter and cheaper paths to high reliability." Binder discusses how testing can help reveal faults in software, allowing for a more stable product to be designed.

Voas [1] introduced the notion of software testability, a quality describing how easy it is to test software. He described information loss issues that lead to loss of testability and introduced a new dynamic analysis metric, *sensitivity analysis*, for determining code's testability. Testability assumes that the tester has a fixed *user profile*, the probability distribution defining the input domain. The implication of this dependency is that the testability of a program varies with the inputs that testers select. Our approach to measuring testability is orthogonal to Voas's sensitivity analysis because we look at the requirement instead of the code.

Much work has focused on using artificial intelligence and machine learning techniques to solve software engineering problems. Here we only list some of them. Gondra proposed an approach for using machine learning to select the software metrics that indicate fault-proneness [17]. Gondra used data on software metric values to train an artificial neural network, and then analyzed this neural network to identify metrics most likely to be useful. Gyimothy, Rudolf, and Istvan presented a study which measures the characteristics of the source code of open source projects [18]. In this study Gyimothy et al. used machine learning techniques to analyze the data obtained from a bug database and showed that such data is useful for fault-proneness prediction. Perini, Angelo, and Paolo proposed an approach for deciding software requirement priority using machine learning [19]. This approach takes into account both the requirements ordering generated using machine learning techniques and the stakeholders' preference. The combination of both types of information facilitates the task of requirements prioritization.

Hayes, Li, and Mohammad introduced a method to facilitate the task of analyzing software requirements using Weka [20]. The key idea of this method is integrating Weka methods into requirement research tools such as TraceLab [21] so that researchers can focus on software requirement analysis. Huang, Raffaella, Zou, and Solc described an approach to detect and classify non-functional requirements automatically [22]. The approach iteratively trains a classifier of non-functional requirements. Sultanov and Hayes introduced a method that uses the machine learning technique Reinforcement Learning (RL) to perform requirement tracing [23]. The study shows that RL generates high quality candidate links between requirement artifacts. Our approach is similar to the above in that we used machine learning to address a requirement engineering and software engineering issue. In contrast, none of the aforementioned studies address our focus: assessing and predicting testability of requirements.

VI. Conclusions and Future Work

We have presented an approach for assessing testability of requirements based on a collection of requirement quality measures. We have conducted an empirical study using a set of statistical and machine learning techniques. Our results suggest that several measures show promise for assessing testability. The measure *now* has moderate correlation to testability for iTrust. The measure *nocw* has moderate correlation to testability for Browser. For both datasets, *fkre* and *fkgl* have lower correlation coefficients, but are much higher than other measures: *cli*, *fkre*, and *pred*. Specifically, we found that *cli* was the most predictive measure in the LR analysis; other statistically significant measures in LR analysis were *nocw*, *pred*, and *fkre*. We found that *fkre* and *cli* were important measures in supervised learning. We also found anecdotal evidence that a model trained on one or more datasets can be used to evaluate requirements from a different, non-trained dataset.

In the future, we intend to perform more extensive empirical studies by considering more real world datasets. We also intend to consider more requirement measures, such as traceability and completeness. Finally, we intend to perform user studies to evaluate how well our approach can help developers improve the testability of requirements.

Acknowledgment

This work has been partially funded by NSF under grant CCF-1511117.

References

[1] J. M. Voas, “PIE: A Dynamic Failure-Based Technique,” *IEEE Trans Softw Eng*, vol. 18, no. 8, pp. 717–727, Aug. 1992.

[2] N. Nagappan, L. Williams, J. Osborne, M. Vouk, and P. Abrahamsson, “Providing test quality feedback using static source code and automatic test suite metrics,” in *16th IEEE International Symposium on Software Reliability Engineering, 2005. ISSRE 2005*, 2005, p. 10 pp.–94.

[3] S. Hesari, R. Behjati, and T. Yue, “Towards a systematic requirement-based test generation framework: Industrial challenges and needs,” in *Requirements Engineering Conference (RE), 2013 21st IEEE International*, 2013, pp. 261–266.

[4] *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)*. Sweden, 2014.

[5] R. V. Binder, “Design for Testability in Object-oriented Systems,” *Commun ACM*, vol. 37, no. 9, pp. 87–101, Sep. 1994.

[6] Department of Defense, *DOD-STD-2167A, Military Standard: Defense System Software Development*. Washington, DC, USA, 1988.

[7] IEEE, *IEEE Standard 610.12-90 IEEE Standard Glossary of Software Engineering Terminology*. IEEE, 1990.

[8] L. Williams, T. Xie, and A. Meneely, *The iTrust Medical Records System*. 2008.

[9] “Readability,” *Wikipedia, the free encyclopedia*. 05-Jun-2015.

[10] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[11]*The Readability Test Tool*. 2015.

[12] D. Gildea and D. Jurafsky, “Automatic Labeling of Semantic Roles,” *Comput. Linguist*, vol. 28, no. 3, pp. 245–288, Sep. 2002.

[13] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, “Natural Language Processing (Almost) from Scratch,” *J Mach Learn Res*, vol. 12, pp. 2493–2537, Nov. 2011.

[14] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA Data Mining Software: An Update,” *SIGKDD Explor Newsl*, vol. 11, no. 1, pp. 10–18, Nov. 2009.

[15] Davis, A.; Overmyer, S.; Jordan, K.; Caruso, J.; Dandashi, F.; Dinh, A.; Kincaid, G.; Ledeboer, G.; Reynolds, P.; Sitaram, P.; Ta, A.; Theofanos, M., "Identifying and measuring quality in a software requirements specification," *Software Metrics Symposium, 1993. Proceedings*, First International, vol., no., pp.141,152, 21-22 May 1993

[16] Boehm, Barry, *Guidelines for Verifying and Validating Software Requirements and Design Specifications*, vol. Euro IFIP 79. North Holland, 1979.

[17] Gondra, Iker. "Applying machine learning to software fault-proneness prediction." *Journal of Systems and Software* 81, no. 2 (2008): 186-195.

[18] Gyimothy, Tibor, Rudolf Ferenc, and Istvan Siket. "Empirical validation of object-oriented metrics on open source software for fault prediction." *Software Engineering, IEEE Transactions* on 31, no. 10 (2005): 897-910.

[19] Perini, Anna, Angelo Susi, and Paolo Avesani. "A machine learning approach to software requirements prioritization." *Software Engineering, IEEE Transactions* on 39, no. 4 (2013): 445-461.

[20] Hayes, Jane Huffman, Wenbin Li, and Mohammad Rahimi. "Weka meets TraceLab: Toward convenient classification: Machine learning for requirements engineering problems: A position paper." In *Artificial Intelligence for Requirements Engineering (AIRE)*, 2014 IEEE 1st International Workshop on, pp. 9-12. IEEE, 2014.

[21] Cleland-Huang, Jane, Adam Czauderna, Alex Dekhtyar, Olly Gotel, Jane Huffman Hayes, Ed Keenan, Greg Leach et al. "Grand challenges, benchmarks, and TraceLab: developing infrastructure for the software traceability research community." In *Proceedings of the 6th international workshop on traceability in emerging forms of software engineering*, pp. 17-23. ACM, 2011.

[22] Cleland-Huang, Jane, Raffaella Settimi, Xuchang Zou, and Peter Solc. "Automated classification of non-functional requirements." *Requirements Engineering* 12, no. 2 (2007): 103-120.

[23] Sultanov, Hakim, and Jane Huffman Hayes. "Application of reinforcement learning to requirements engineering: requirements tracing." In *Requirements Engineering Conference (RE)*, 2013 21st IEEE International, pp. 52-61. IEEE, 2013.

Appendix

TABLE 2. BROWSER REQUIREMENTS STATISTICS

	fkre	fkgl	cli	now	nocw	num preds
Mean	64.9	9.0	10.4	44.0	3.5	7.0
Median	67.1	7.6	10.1	38.0	4.0	6.0
Min	33.7	4.2	7.8	15.0	0	1.0
Max	82.5	13.9	15.2	77.0	7.0	17.0
Variance	242.7	12.6	4.5	456.4	7.7	24.0

TABLE 3. ITRUST REQUIREMENTS STATISTICS

	fkre	fkgl	cli	now	nocw	num preds
Mean	48.3	12.2	12.1	73.8	9.6	7.9
Median	47.7	11.7	12.0	67.0	8.0	6.5
Min	14.2	6.0	6.4	20.0	0	2.0
Max	83.6	24.4	16.7	197.0	32.0	21.0
Variance	222.3	14.4	4.9	2305.4	52.6	26.2

Browser "best" : R8: "The html display zone shall be located directly under the toolbar/address field. It shall have the width of the browser window and shall occupy the rest of the height of the browser window (except for the small space needed to divide it from the toolbar/address field and any space you want to allocate for the status bar at the bottom)."

AGL	FKRE	FKGL	GFS	SMOG	CLI	ARI
13	55.1	13	16	10.1	10.9	14.9

Browser "worst" : R9: "The type and the number of controls you decide to use for the html display zone is not fixed by the requirements. Your solution must satisfy all requirements set in this document, but otherwise is up to you."

AGL	FKRE	FKGL	GFS	SMOG	CLI	ARI
10	62.9	9.2	13.9	10.1	10.1	8.8

Figure 1. Example of "Best" and "Worst" Requirements and Measures

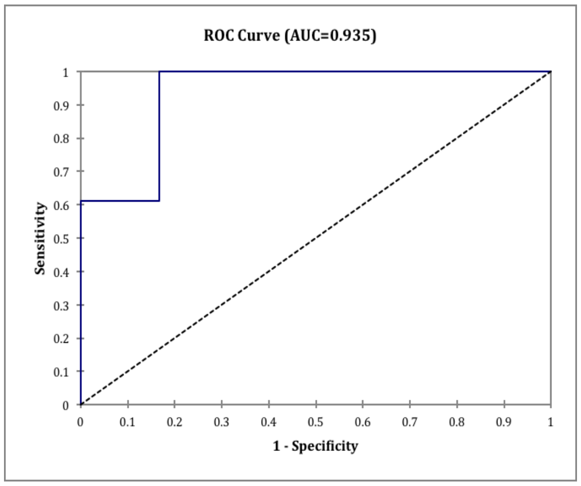

Figure 2. ROC curve for combined measures for iTurst dataset

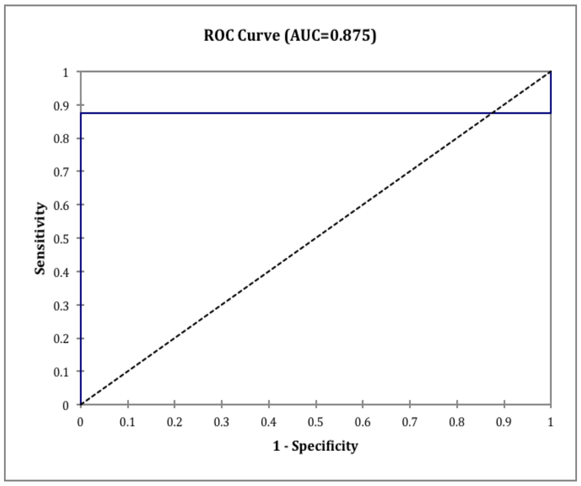

Figure 3. ROC curve for combined measures for Browser dataset