A Simpler Model of Software Readability

Daryl Posnett
University of California, Davis
Davis, CA
dpposnett@ucdavis.edu

Abram Hindle
University of California, Davis
Davis, CA
ah@softwareprocess.es

Premkumar Devanbu
University of California, Davis
Davis, CA
devanbu@ucdavis.edu

ABSTRACT

Software readability is a property that influences how easily a given piece of code can be read and understood. Since readability can affect maintainability, quality, etc., programmers are very concerned about the readability of code. If automatic readability checkers could be built, they could be integrated into development tool-chains, and thus continually inform developers about the readability level of the code. Unfortunately, readability is a subjective code property, and not amenable to direct automated measurement. In a recently published study, Buse *et al.* asked 100 participants to rate code snippets by readability, yielding arguably reliable mean readability scores of each snippet; they then built a fairly complex predictive model for these mean scores using a large, diverse set of directly measurable source code properties. We build on this work: we present a simple, intuitive theory of readability, based on size and code entropy, and show how this theory leads to a much sparser, yet statistically significant, model of the mean readability scores produced in Buse's studies. Our model uses well-known size metrics and Halstead metrics, which are easily extracted using a variety of tools. We argue that this approach provides a more theoretically well-founded, practically usable, approach to readability measurement.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Human Factors, Theory, Measurement

Keywords

Readability, Halstead, Entropy, Replication

1. INTRODUCTION

Readability of code is of central concern for developers [1, 18, 19, 23]. Code that is readable is often considered more maintainable; code that is more readable today is presumed to remain easier to read, comprehend, and maintain at a later date.

We conceive of readability as a subjective impression that programmers have of the difficulty of code, as they try to understand it. The relationship between readability and understanding is analogous to syntactic and semantic analysis, readability is the syntactic aspect while understandability is the semantic aspect. In essence, readability is a perceived barrier to understanding that the programmer feels the need to overcome before working with a body of code: the more readable it is, the lower the barrier.

There is much previous work about readability [7, 15, 33, 34, 13, 3]. One major issue with studies of readability is the difficulty of experimentally reifying what is essentially a subjective perception. Measures of subjective perception are both difficult to obtain, requiring human studies, and also are inherently variable; large-scale surveys, involving multiple human raters, and careful statistical analysis of inter-rater agreement are required to obtain usable measures. Buse *et al.*'s work was a major contribution in this area: they conducted a fairly large-scale study, asking human subjects to provide subjective ratings of the readability of code snippets. These scores were validated and aggregated to yield mean-opinion-scores [6]. The result of this extensive and time-consuming study was a set of code-snippets, $\mathcal{S}$, accompanied by mean subjective readability scores: $\mathcal{O}(s), \ s \in \ \mathcal{S}$. Buse *et al.* then gathered direct, automatically-derived, token-level measures, $\mathcal{M}(s)$, of the code snippets. They then built a logistic regression model to predict the subjective, laboriously gathered scores, $\mathcal{O}(s)$, using the collection of automatically gathered direct metrics, $\mathcal{M}(s)$. The $\mathcal{M}(s)$ were essentially token-level measures, but even so, were able to predict $\mathcal{O}(s)$ to some degree. This was an important contribution since it opens up the possibility of automatic tools that can provide readability scores as feedback to developers for every commit they make; this continuous feedback might potentially improve readability, and thus maintainability, of code over time.

In this paper we improve upon Buse *et al.*'s model, yielding a model that is *simpler, better performing* and *theoretically well-founded* in both classical software engineering and basic information theory. In particular we argue that fundamental aspects of readability have been measured since the 1970s by Halstead's software science metrics [21]. We show that the Halstead metrics can be used to improve upon the

Avg.	Max.	Feature
●	●	identifier length
●	●	indentation
●	●	line length in characters
●		# blank lines
●		# spaces
●		# comments
●	●	# numbers
●	●	# identifiers
●	●	# keywords
●		# assignments
●		# branches
●		# loops
●		# arithmetic operators
●		# comparison operators
●		# parentheses
●		# periods
	●	# occurrences of any single character
●	●	# occurrences of any single identifier

Table 1: Token features captured by Buse *et al.* and reported in [6].

Buse *et al.* model of mean-opinion scores of snippet readability. In addition, we introduce some simple, intuitive measures of entropy into our models, and show that these measures further improve model performance. In the ensuing discussion, we use *Buse's model* to refer to the Buse *et al.* model, and the *improved Halstead model* for the model introduced in this paper. Like Buse's model, the improved Halstead model is based on lexical properties, and can be gathered quickly and easily with widely-available tools. Furthermore, it can be gathered from inchoate code that is not yet compilable.

Below, we begin with motivation, theoretical foundations, and our research questions; we then present our methodology. After the results are presented, we conclude with warnings and speculations concerning the further implications of this work.

2. CONCEPTUAL FOUNDATIONS AND RESEARCH QUESTIONS

We build on the Buse Model, aiming to make it more robust, theoretically well-founded and usable. Specifically, we were animated by 3 goals:

1. Simplify the model, and improve its performance.
2. Clarify and strengthen its theoretical bases in the classical software engineering ideas of size and complexity.
3. Further improve the model by using simple information-theoretic notions of entropy.

Our approach to these goals is described below.

2.1 Statistical Modeling

At the core of the Buse *et al.* work is a statistical prediction model, which regresses the subject $\mathcal{O}(s)$ measures against the direct lexical $\mathcal{M}(s)$ measures. There are various recommended guidelines to ensure the robustness and validity of estimated models.

One rule of thumb when building prediction models is that one should have 10 to 20 times the number of samples as predictor variables [32]. Models built with insufficient cases relative to the number of predictors can be unstable and often, due to feature overlap, tell us very little about what features have genuine and significant impact on the result.

Buse *et al.* had a sample of 100 mean subjective ratings. The above rule of thumb suggests that about 5-10 predictors could be used with this sample size to avoid instability due to over-fitting. However, they used (in aggregated or extremal summary) 19 different lexical metrics to yield 25 different features. Buse *et al.* also reported that principal components analysis *(PCA)* revealed significant overlap in their feature set and that 95% of the total variance could be explained by only 6 principal components. This suggests not only that some features do not help, but also that a simpler model may lead to a more general theory of readability. We begin with a simple, yet general research question:

Research Question 1: Is there a simpler explanation for the mean readability scores gathered by Buse *et al.* that does not depend on a large number of features?

2.2 Size and Readability

It is a reasonable assumption to believe that size should not affect a measure of readability; *viz.* if two functions are equally readable then why should their concatenation be less readable? However, if taken too far, this argument might lead to the absurd conclusion that a single class, method, statement, or even token, of a program should have the same measure of readability as the entire program. In Buse's study, the snippets presented to the raters vary in size; so it is quite possible that variation in size has some effect on the readability ratings. If the experimental design does not control for the effect of size explicitly, and we wish to develop hypotheses that relate code attributes to readability, then it is necessary to include such control in the model [14, 5].

Buse *et al.* argue that they avoid the issue by choosing features believed to be *"independent of of the size of the code block"* ([6], §4.1, para 1; by "code block" here they mean the snippets handed out to raters for evaluation). Their expectation is that this approach would yield a model independent of size. We argue, however, one must explicitly include size in a model in order to discern size dependency from the effect of non-size factors.

In support of this position, we display the Spearman correlation that several of the features used by Buse *et al.* have with size measures (lines, words and characters, as measured with UNIX `wc` command) in Table 1. The features clearly are not independent of size.

Metrics	lines	words	characters
avg math	0.45	**0.66**	0.51
avg comment	0.57	**0.63**	0.49
max idents	0.30	**0.66**	**0.65**
max word	0.34	0.49	0.54
max line length	0.11	0.44	**0.62**
max occurrences char	0.47	**0.62**	**0.84**

The feature *max occurrences of a single character*, for example, is sufficiently correlated with the number of characters that it could potentially be viewed as a proxy for absolute size of the snippet. This leads to our next research question:

	Lines	Characters	N	n	V	D	E	Token Entropy	Unique Tokens	Byte Entropy	Voter Mean
Lines	1.00	0.69	0.18	0.24	0.21	0.16	0.15	-0.19	0.28	-0.04	0.23
Characters	0.69	1.00	0.57	0.51	0.57	0.18	0.30	-0.56	0.51	0.06	-0.20
N	0.18	0.57	1.00	0.89	0.99	0.56	0.75	-0.94	0.87	-0.01	-0.64
n	0.24	0.51	0.89	1.00	0.93	0.62	0.74	-0.81	0.99	-0.04	-0.61
V	0.21	0.57	0.99	0.93	1.00	0.59	0.77	-0.92	0.91	-0.02	-0.64
D	0.16	0.18	0.56	0.62	0.59	1.00	0.95	-0.50	0.64	0.00	-0.40
E	0.15	0.30	0.75	0.74	0.77	0.95	1.00	-0.69	0.75	-0.01	-0.50
Token Entropy	-0.19	-0.56	-0.94	-0.81	-0.92	-0.50	-0.69	1.00	-0.79	0.01	0.62
Unique Tokens	0.28	0.51	0.87	0.99	0.91	0.64	0.75	-0.79	1.00	-0.05	-0.57
Byte Entropy	-0.04	0.06	-0.01	-0.04	-0.02	0.00	-0.01	0.01	-0.05	1.00	-0.10
Voter Mean	0.23	-0.20	-0.64	-0.61	-0.64	-0.40	-0.50	0.62	-0.57	-0.10	1.00

Table 2: Spearman correlation of Halstead metrics, Size, Entropy measures, and mean voter opinion score.

Research Question 2: Can the mean readability scores be explained simply by the size of the snippets?

The most commonly used metric for size of code, LOC [1], however, does not correlate with readability scores, as much as the number of words and characters. Further, our intuition about readability suggests that there is more to readability than just how much code that we must read, but that it also depends on the content. The Buse *et al.* metrics measure content by measuring the mean and max of various syntactical attributes of the code.

We observe that many of these metrics are, in essence, token class counts, e.g. *#parenthesis*, *#commas*, *#loops*; this raises the question, whether these features could be subsumed by a measure that incorporates token class diversity. This approach is additionally motivated by observing that several of Buse's top performing individual metrics, in terms of correlation with the readability scores, are counts of identifiers and keywords. This reasoning led us back to Halstead's work.

2.3 Halstead's Metrics

Maurice Howard Halstead introduced statically computed source code complexity metrics in 1977 [21]. These purely lexical measures are calculated from counts of the number of total operators, unique operators, total operands and unique operands. Operators include methods, while operands are the participants in the operation, such as method arguments. Then these four measures of operators and operands are combined together into various Halstead metrics such as:

Program Length is the sum of the total number of operators (N_1) and operands (N_2). $N = N_1 + N_2$.

Program Vocabulary is the sum of the number of unique operators (n_1) and unique operands (n_2). $n = n_1 + n_2$

Volume is the program length (N) times the log_2 of the program vocabulary (n). $V = Nlog_2 n$. This measure is similar to entropy, but represents the minimum number of bits needed to naively represent the program.

Difficulty is half of the unique operators multiplied by the total number of operands, divided by the number of distinct operators. $D = \frac{n_1}{2}\frac{N_2}{n_2}$

[1]Here we are using snippet lines which is not precisely lines of code as some snippets contain comments

Effort is the difficulty multiplied by the volume. $E = DV$. Effort was intended as a suggestion for how long code review might take.

The *volume* Halstead measure arguably aims to measure the information content of the source code, by combining total counts with unique counts. One can picture a reader poring through the code, encountering new operators and operands and being surprised by and trying to decipher each: at each one, they pause and look up the operator or operand in their mental "symbol table". Thus a close relationship to readability can be hypothesized:

Research Question 3: Do the Halstead metrics add additional explanatory power to the size readability model?

This discussion suggests an even simpler view: is readability simply related to the information theoretic notion of entropy.

2.4 Entropy

Entropy is often viewed as the complexity, the degree of disorder, or the amount of information in a signal or data set.

Entropy is calculated from the counts of terms (tokens or bytes) as well as the number of unique terms and bytes. Where X is a document and x_i is a term in X. $Count(x_i)$ is the number of occurrences of x_i in the document X and $p(x_i) = \frac{count(x_i)}{\sum_{j=1}^{n} count(x_j)}$. Entropy $H(X)$ is defined as follows:

$$H(X) = -\sum_{i=1}^{n} p(x_i) log_2 p(x_i)$$

The *terms* here can be bytes or tokens, and we use both in this paper.

While Halstead's *volume* looks superficially similar to entropy, the calculation is quite different. Entropy calculations depend on the relative distribution of the tokens/characters in the code body under consideration, with uniform distributions giving the highest entropy, and highly skewed distributions yielding lower entropy; whereas volume attempts to determine the number of bits needed to represent all operators and operands multiplied by the total number of tokens.

Our next two research questions relate to the notion of entropy. First, we consider the effect of adding byte level entropy to our model.

Research Question 4: Does byte entropy contribute to the readability model?

Despite the conceptual and mathematical differences between token entropy and Halstead's V we cannot ignore their very strong inverse correlation. Consequently, we also want to know if token-level entropy measures are equivalent to Halstead's V in our prediction model.

Research Question 5: Can Halstead's V be replaced with token entropy in our readability models?

Finally, we recognize that any model of readability is of limited usefulness if it does not scale up in a reasonable way to larger fragments. We consider how a model of readability, learned on a small sample of snippets, can be used to capture readability of larger source code elements. Unfortunately, we do not have additional human-scored large code elements. We therefore try a simple test: does the model score some larger elements as readable, and others as not? Or does it rate all large elements one way or the other. An *a priori* belief that all large code elements are equally readable (or not) seems unjustifiable; so we must be doubtful of a model that rates all large code elements the same way. While this test does not prove that a model that discriminates is discriminating correctly, one can conclude from the test that a model that classifies all large files as readable (or not) lacks credibility.

Research Question 6: Does our simplified model generate varying readability scores for larger code fragments?

In the following sections we describe our data gathering process and model building methodology, our evaluation of our readability metric, related work, and some threats to the validity of our study.

3. DATA & STATISTICAL MODELING

3.1 Readability Data

The Buse readability survey is a publicly available trove of data containing $12,000$ human judgements by 120 annotators on 100 snippets of code.[2] The annotators were computer science students in various stages of study, from 1st year university to graduate student. Each annotator rated code snippets on a scale from 1 (low readability) to 5 (high readability). The average of these scores across all 120 annotators are computed for each snippet. Using these mean opinion scores for the 100 snippets, we label each snippet as *more* or *less* readable using the same score threshold of 3.14 used by Buse *et al.* We also compute size metrics, Halstead metrics, and token and byte level entropy, for each of the snippets and use this data to train classifiers using the Weka toolkit[20]. Using publicly available source code [2] provided by the author, we replicate the models reported in [6]. We use 10-fold cross validation to avoid over fitting our model to the training data and repeat this 10 times with different seeds to help correct for bias introduced by each classification method. Since the number of instances in each class is unbalanced we use stratified sampling and compute the weighted mean of each performance measure to reflect this imbalance.

[2]http://www.arrestedcomputing.com/readability/

3.2 Model Building

When building a model with many predictors, there is a challenge of choosing the right set of predictors to include in the model so that the model is statistically significant, parsimonious, and provides a good fit for the data. Backwards stepwise refinement is a model building technique whereby a classifier is allowed to choose from many features. It is a, typically mechanized, feature selection process to remove features that do not improve the model. The practice is commonly used and is sometimes acceptable in the context of a prediction model. This approach, however, can fail to find the best model for any number of reasons. There may be more than one set of features that explains the data, or, interaction between groups of variables may cause the procedure to eliminate variables that should be in the model. Buse *et al.* use an automated feature selection approach; so while model performance is satisfactory, it is difficult to draw any conclusions about the impact of the variables used in the model.

Forward stepwise refinement is the process of starting with minimal models and *manually* adding variables, and gauging their impact on the quality of the model. While this process has its critics (as its success depends on the order of adding variables), it is usually preferred when building a model; specially when building a model to test hypotheses [9]. We use this approach to shed light on the research questions outlined above.

3.3 Model Evaluation

We draw from a set of performance measures to evaluate models. It should be noted that all models use direct lexical-based automated measures to predict readability classification (more or less readable); we then compare the predicted classification with the human-based classification from the Buse data.

F Measure The F-Measure is the commonly-used harmonic mean of precision and recall. Classifiers output a probability that must then be thresholded to yield a classification decision. Lessmann *et al.* argue that the requirement of defining a threshold is reason enough not to use such simple static measures in a prediction context; however, we include it as it is well understood by the community [27].

Percentage Correct The percentage correct is a well understood measure which, although intuitive, is also dependent on choosing a cutoff value.

ROC An established method of evaluating classifiers independently of any particular threshold is *Receiver Operating Characteristic* (ROC) analysis. A ROC curve represents a family of precision/recall pairs generated from varying the threshold value between 0 and 1 and plotting the False Positive Rate $FPR = \frac{FP}{FP+TN}$ on the x-axis and the True Positive Rate $TPR = \frac{TP}{TP+FN}$ on the y-axis. All such curves pass through the points $(0,0)$ and $(1,1)$. The point $(0,1)$

represents perfect classification and points on the ROC curve close to $(0, 1)$ represent high quality classifiers. A common way to evaluate the overall quality of the classifier is to compute the area beneath its ROC curve.

4. EVALUATION

We consider each of the research questions in turn. The first hypothesis concerns essentially the parsimony of an effective model. We later present a simple model that includes many fewer different predictors than the Buse model; so we reserve that discussion until we have presented the results leading up to that model.

4.1 Size and Readability

If we believe that size may have influenced the readability scores, then the trivial hypothesis is that the readability of a snippet is primarily a function of size. Intuitively we do not expect that this model will work well, but that it should provide a baseline measure of the size dependence within the snippets. We build a classification model based simply on the number of lines in the snippet and then extend this model to include both the number of words, and the number of characters. To guide model construction we begin with the Spearman correlation between mean voter scores and size metrics.

Metrics	lines	words	characters
mean voter score	0.232	−0.002	−0.202

The low correlation between each metric individually and the mean voter scores clearly indicates that no single size metric is likely to yield a good prediction model. Positive correlation between the number of lines and the mean voter scores suggests that snippets with more lines are more readable and, similarly, negative correlation between characters and lines suggests that snippets with a large number of characters, *i.e.* larger file size, are less readable. Given the low correlation between words and mean voter scores we build two classification models, one with lines and characters, and the other with all three metrics.

For each of the 10 classifiers we obtain the mean of the 10 runs over the performance metrics described in the previous section. The reported value is the mean performance of all classifiers over all runs.

Model	% Correct	F-Measure	ROC
Token Features (Buse)	75	0.74	0.77
Size: Lines,Char	63	0.60	0.64
Size: Lines,Char,Words	65	0.62	0.65

A one-sided Wilcox test on the ROC results verifies that a model based on the Buse feature set outperforms the simple size model with p-value = 0.00447. A two-sided Wilcox test verifies that there is no difference between the two size models with p-value = 0.796. So, with respect to RQ2:

Result 2: *Size alone does not explain the reported mean readability scores.*

We note here that the high correlation between lines and characters might (see Table 2) lead to multicollinearity. The low correlation between any of the size measures and the mean voter score suggests that single parameter models would not likely perform well. Had this model demonstrated excellent performance, we would have more seriously considered multicollinearity issues between lines and characters before adding more variables.

Although size does not fully explain readability, in contrast to the approach taken by Buse *et al.*, we can see that *size is a significant predictor of readability and should be included in any model that seeks to understand what factors affect readability.* Moreover, we observe that lines and characters have opposite effects, but words, *e.g.* tokens and identifiers, have a negligible effect when considering only their counts in a model that includes size.

4.2 Halstead and Readability

It is well known that many of Halstead's metrics are highly correlated to themselves; if used together in any multivariate model they will exhibit significant multicollinearity [24]. It is, therefore, necessary to consider the correlation among our potential predictors [9]. Table 2 confirms that N, n, and V are highly correlated, suggesting that including more than one in a classification model is unlikely to significantly improve the model.

There is also a fairly high degree of correlation between the Halstead metrics and the mean human rating. While D and E have reasonably high correlation to the other Halstead metrics, and also to mean human rating, their interpretation has been criticized due to Halstead's misapplication of results from cognitive psychology [11]. All of the Halstead metrics are negatively correlated with the mean voter scores. Also interesting, as in our previous model, the number of lines in the snippet is positively associated with readability.

We consider first the Halstead metrics alone in a logistic regression model. We find that multicollinearity is excessive and that, as expected, we can have at most, one of N, V, or n. Any use of these variables alone performs comparably to the Buse feature set. Proceeding in a forward stepwise refinement we add both D and E in turn to the model. We find that neither variable contributes significantly to the model and consider them no further. Both characters and lines represent the size of a snippet but are correlated differently with the Halstead metrics and with the mean voter scores. Characters have both a high correlation with lines and with V, and, like V, are negatively correlated with the mean voter scores. This suggests that when both characters and lines are added to a model that includes Halstead metrics, one or both may lose significance.

Adding the number of lines in the snippet to the model improves the fit of the model. As with the correlation, the coefficient for V is negatively associated and the number of lines is positively associated with mean readability and the two variables alone explain approximately one half of the variance in mean readability. This result suggests the following: First, as either the number of tokens, or, the number of unique tokens increases, readability decreases as the size remains constant. Second, if we increase the size of the snippet, *i.e.* we provide a greater area for the existing tokens to spread out, readability increases. Adding characters to the model does increase the fit somewhat, but with dramatically lower significance and, therefore, we remove it from the model. The performance of this model as compared to the Buse model is presented in Figure 1. The *simple, parsimonious model with two predictors, size and V, outperforms the much richer Buse model* across all performance measures.

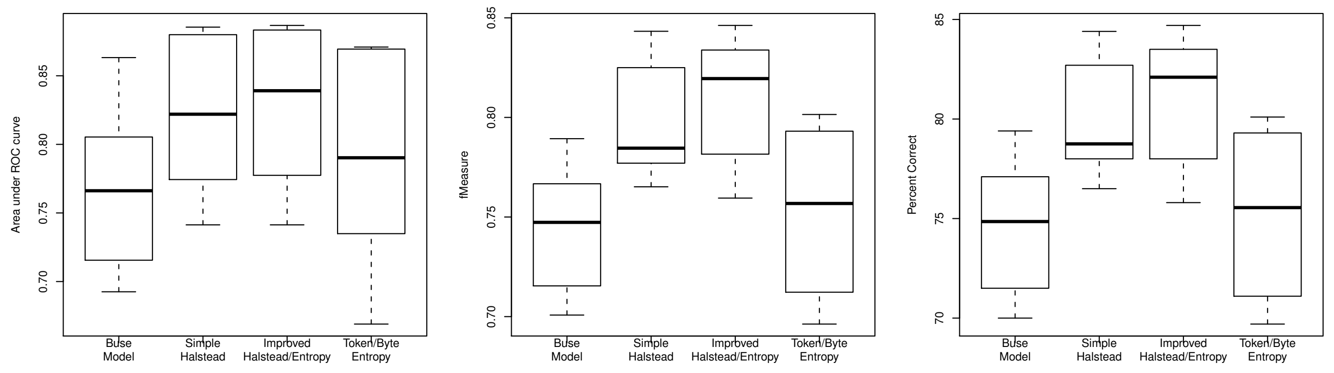

Figure 1: *Buse* is the model reported by Buse *et al.* *Simple Halstead* includes only lines and V. *Improved Halstead* includes lines, V, and character entropy. *Token/Byte Entropy* includes lines, token entropy, and byte entropy. Performance of 10 classifiers on the Buse *et al.* token feature models and the simpler Halstead/Entropy models using the means of 10 runs of 10-fold cross validation. A one sided Wilcox signed rank test shows the Simple Halstead measures greater than Token Count with p-value = 0.02621 for ROC, p-value = 0.000525 for f Measure, and p-value = 0.0007523 for percent correctly classified. Similarly, the improved Halstead measures are greater than Buse's Token Feature with p-value = 0.01440 for ROC, p-value = 0.0002436 for f Measure, and p-value = 0.0002436 for percent correctly classified. The token/byte entropy model is not statistically better, or worse, than Buse's token feature model.

A snippet that contains many distinct elements, both operators and operands, will have greater volume than a snippet with a similar number of total elements but fewer distinct elements. Our results indicate that element identity is less important than element cardinality and diversity. Similarly, the number of lines in a snippet is positively associated with readability. When a programmer views a snippet of code, the dimensions of the snippet have some importance. It is possible that two snippets with identical Halstead V could have considerably different character counts due to a difference in identifier or keyword lengths. The number of lines captures this dimension of readability.

Result 3: *Halstead's V adds considerable explanatory power to the models and, when combined with a simple size measure, outperforms the Buse model.*

4.3 Entropy Measures

As discussed in Section 2.4, we studied whether the pure information content of the snippet, *i.e.* its byte level entropy, might affect readability. The raw correlation between entropy and readability is low and negative, suggesting that an increase in raw information content will actually reduce readability somewhat. Adding the byte level entropy to the "simple Halstead model", which just includes size and V, gives the "improved Halstead model", which shows somewhat improved performance. The performance of this model as compared to the Buse model is also presented in Figure 1. The improved Halstead model (with byte entropy) outperforms the Buse model across all performance measures, and somewhat improves the performance of the simple Halstead model.

Result 4: *Although byte entropy improves the prediction model, its impact on performance is small compared to the other predictors.*

As discussed previously, Halstead's V relies on token counts and frequencies. Therefore it is reasonable to consider the effect of *token*-level entropy measures. We computed the entropy of Java tokens in the snippets and compared models using this metric to metrics using Halstead's V. Using token entropy in place of V yields models whose performance is comparable to the Buse model; except that the token entropy measures have a positive coefficient. The larger positive (rather than negative, as with V) correlation between token entropy and mean human rating is simply a consequence of the inverse relationship between the Halstead's V and token entropy. Token entropy uses a relative frequency calculation, with the total token count in the denominator, whereas V has this figure in the numerator. As can be seen in Table 2 token entropy and V are negatively correlated.

Result 5: *Token level entropy explains the data as well as the Buse token feature models but not as well as Halstead.*

4.4 Snippet Size, Methods and Classes

The Buse human ratings were done entirely with small code snippets. All of the annotated snippets range from 4 to 11 lines, including comments, and do not span function boundaries. There are no class definitions and not even an entire function included within these snippets. The question naturally arises, does the Buse *et al.* readability model scale up to large code fragments? What happens when we attempt to use this model for entire functions, classes, etc.?

To begin to address this question, we extracted functions from Lucene 2.4.We compute the readability scores using our model and the Logistic model reported by Buse *et al.* The scatter plots of classifier probability against lines of code are shown in Figure 2. Buse's Logistic model ranks all functions longer than about 200 lines as less readable. The improved Halstead model presented above continues, as number of lines increases above 200 lines, to classify some functions as readable, and others as not. We argue that it is not reasonable to classify *all* functions longer than 200 lines as uniformly less readable; thus we also argue the behaviour of the improved Halstead model is more reasonable, *a priori*, than the Buse *et al.* model. However, we cannot make strong claims to greater validity here. The improved Halstead model was trained on the same data and may very well fail to classify correctly at a larger size. Readable functions may be classified unreadable and vice versa. However, it is quite unlikely that all large functions in Lucene are unreadable and so that it is even less likely that Buse's token feature models are valid at that size.

We can see also an interesting bifurcation (see the second and fourth figures from the top in Figure 2) in the improved Halstead model: beyond a certain size, both class and functions are either very readable, or very unreadable. If we are using these models correctly, *i.e.* as a classifier, however, so long as the direction is correct, this is less problematic, and our model continues to discriminate between readable and unreadable functions.

When we apply these metrics to entire classes we observe a similar loss of discriminatory power beyond a certain size. Buse's models classify only two classes as readable beyond about 50 lines and the improved Halstead model classifies only a single class as readable beyond about 250 lines. Above 250 lines, the improved Halstead model classifies *only* `IndexModifier` in package `org.apache.lucene.index` as readable; above 50 lines, Buse's model classifies `SynExpand` in package `org.apache.lucene.wordnet` as well as `BitUtil` in package `org.apache.lucene.util` as readable. We conclude that both models have limited size validity but that the Halstead model has discriminatory power over a larger range.

Result 6: *The improved Halstead model shows discriminatory power over a larger size range; however, lacking human ratings, we cannot validate the readability classifications produced by these models over larger sized entities, including entire functions and classes.*

4.5 A Simpler Model of Readability

We return now to the first research question. *How complex does a readability model have to be, in order to be useful?* The predictive metrics included in all the models discussed are easy to extract. However, a more parsimonious model is easier to understand and interpret from a theoretical perspective. Also, simpler models are more *actionable*: programmers can more easily respond to simple instructions on how to improve readability.

Our results show that with only 3 variables our model outperforms the Buse model as a classifier of readability in small code snippets. Using the logit function $\frac{1}{1+e^{-z}}$ we obtain a simple model using the following expression for z:

$$z = 8.87 - 0.033\ V + 0.40\ Lines - 1.5\ Entropy$$

We do not assert that this simple model captures the essence of readability nor that it can be applied blindly to any piece of code. Rather, we observe that this model fits the gathered data reasonably well, and, with significantly fewer variables, is a more parsimonious model of readability within the limited context studied.

We test the performance of our choice of features using ten of the classifiers available in Weka and also supported in the RWeka package.[3] Figure 1 shows that models built from our smaller feature set compare favourably in performance to the models built from features presented by Buse *et al.*

Result 1: *The Buse mean readability data can be explained with a much simpler model that includes only three features.*

5. DISCUSSION

While readability is subjective (and thus a matter of some debate), and probably also contextual (depending on the individual reading and the operative task), there is no shortage of assertions that it is beneficial [1, 18, 19, 23]. It should be noted that the Buse readability scores that we used were *not* gathered from developers embarked on a specific task, but from students with no investment in the code. In particular, they were not being assessed on either their ability to *read to recall* or *read to do.*

While this, *per se*, may not necessarily have affected the validity of Buse's scores, it's quite possible that "readability" means something else to a developer who is invested in the code. With that caveat, our models indicate that readability of code strongly depends on the information content in the source code as determined by V and entropy.

5.1 Indentation

A sharp-eyed reader might wonder, does indentation not contribute to readability? Why then does it not figure into these models? Buse's model does incorporate a measure of indentation that will have some correlation with the block structure of code. In fact, indentation variance has been shown to have a high degree of correlation with McCabe's Cyclomatic complexity [22]. We considered several measures of indentation including Buse's measure of maximum indentation, mean of indentation, variance of indentation, and mean and variance of line length. The results of including these measures were mixed. They had a negligible effect on the classification models presented in Figure 1, however, in some cases they did improve the amount of variance explained in *ordinary least squares* (OLS) regression models of the mean opinion scores. Further, when including indentation in a classification model the range of discriminatory power of the model increased, *viz.,* the bifurcation effect in Figure 2 extended somewhat further. All in all, we found that the additional beneficial effect of including indentation measures in our models to be modest.

[3]http://cran.r-project.org/web/packages/RWeka/index.html

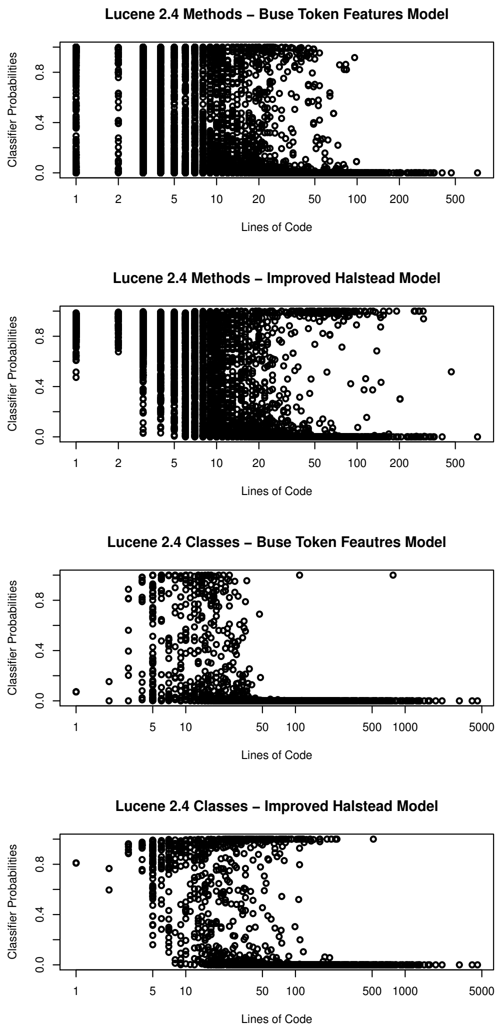

Figure 2: Size versus classifier probability scatterplots of all functions and classes in Lucene 2.4.

5.2 Outliers

We could typically explain about 55% of the variance (R^2) in the mean readability scores when building an OLS regression model with either our features or the token count features. We found, however, a few of the snippets were potential outliers. [4]

When we removed the top 5% of these outliers the models improved dramatically and we were able to explain about 70% of the variance. The goal, of course, is not to eliminate the pesky data, but to understand the data. In other words, about 5% of the data was not reasonably explained by either our features, or the token count features. Consequently, it may be possible to significantly improve the model by identifying what makes these snippets special. One of the shortest snippets, number 6, is shown below. For our models this snippet appears to be an outlier with respect to V for its size, which is quite low compared to other snippets with a similar readability score and size.

```
xsp = jj_scanpos;
if (jj_scan_token(100)) {
jj_scanpos = xsp;
if (jj_scan_token(101)) return true;
```

Of course, observing the attributes that make it an outlier is not the same as determining why it is an outlier, *viz.* why its readability is low given its low V? It is likely that for some unknown reason, neither the features presented by Buse *et al.*, or those presented here, capture the notion of unreadability present in this snippet. It's also possible that such snippets are an artifact of the experimental set up; *e.g.,* perhaps there was systematic experimental issue that caused such snippets to be shown early or late to the subjects, thus influencing the human rating.

6. RELATED WORK

Software *readability, maintainability, understandability* and *complexity* are related, and consequently the terms are often conflated in the literature. Beyond Buse *et al.* [6], much of the relevant previous work measured artifacts at different granularities with different definitions of readability.

Early work often linked readability with comprehension. Baecker *et al.* hypothesized that the appearance of code affects comprehension and conducted two experiments on C source code presented in two typographic formats and found that the relationship between comprehension and readability is partially dependent on the task in question [3]. Détienne *et al.* present a review of further work in this area, and in particular, discuss the difference between the *reading to recall* and *reading to do*; *viz.* the purpose of reading is either to summarize what was read or to later work with the material being read [13].

Studies that focus upon *readability* explicitly, beyond Buse *et al.*, range in methods from comment counting, to English-based readability scores, to identifier readability. In a study on software maintainability [6], Aggarwal *et al.* used the ratio of comments to lines of code as a proxy for readability [2]. This simple proxy is also often used when building software quality models [12, 25]. Börstler proposed a readability metric similar to the Flesch [17] reading score for the English language called SRES [4]. The authors related their score to both Halstead's difficulty and Cyclomatic complexity. Butler *et al.* studied the nature of identifier quality, whether or not an identifier adheres to style guidelines, and their relation to code quality and readability [7, 8].

These studies approached readability in different ways but none of them except Buse *et al.* relied on human ratings of readability [6].

Maintainability models based on Halstead's metrics and other code complexity metrics have been studied for some time and are used in commercial metrics tools. Coleman *et al.* built several models of maintainability and evaluated their metrics by noting that an analysis of maintainability of two commercial subsystems corresponded to human evaluation of the systems [10].

Understandability is a distinct issue that deals with more semantic issues. Lin and Wu integrated employed fuzzy statistics theory to build a comprehensive model of understandability [28, 29].

Complexity has been studied heavily and is often associated with readability and understandability. Chhabra *et al.*

[4]No datapoints were ignored, or otherwise harmed, while training our models.

defined a code spatial complexity understandability metric based on the mean distance between use and definition of a variable, and related this metric to the time to complete a measurable perfective maintenance task [26]. They found that reductions in distance result in reductions of maintenance task completion time. Mohan *et al.* proposed a metric, *COMP*, that also used code spatial complexity, along with simple *typographical style* metrics, *e.g. indentation*, *blank lines*, and concept counts, to assess program comprehensibility in COBOL II programs [30]. Naeem *et al.* presented metrics intended to evaluate the effectiveness of decompilers and obfuscators based on textual and lexical complexity [31]. They used complexity metrics, as well as some token based features similar to Buse *et al.*, and showed that these metrics vary with the degree of obfuscation.

In terms of work that is close to ours, Etzkorn *et al.* define a semantic entropy metric based on keywords and concepts in source code [16]. The keywords and concepts are derived from a source base using heuristics based on the linguistic and structural aspect of object oriented software. A human study demonstrated correlation between their metric and perceived complexity of software. Our approach is different in both desired outcome and complexity. Our metrics are lightweight and do not rely on a generated knowledge base.

7. THREATS TO VALIDITY

In this section we examine threats to each form of validity in our study and the methods used to mitigate these threats where possible.

Construct validity refers to the degree to which measured properties are correlated with the theoretical concepts they are believed to represent. The primary threats to construct validity arise from the limitations present in the study by Buse *et al.* [6]. One major threat is that we rely on a single source of data, Buse *et al.*'s ratings. Fortunately, Buse *et al.*'s ratings rely on multiple sources: 120 student raters producing 12000 ratings of 100 snippets. We assume that raters judge the snippets on perceived readability alone and that there is some common notion of readability that can be captured by a mean score. We also assume that the motivation of the reader does not affect the readability of the snippet. Although we know that the raters were computer science students, without access to the raters we do not know their level of expertise relative to their education level.

Internal validity relates to the validity of causal inferences made by the study. Our study is focused on examining a link between readability and token entropy measures. We believe that our approach, which combines parsimony with stronger theoretical bases, reduces internal validity risk.

It should be noted that we compute entropy and V using only the tokens in each snippet to identify the number of different unique tokens (token vocabulary). It might be argued that the entire set of snippets should be considered together to induce vocabulary. However, each rater only saw a subset of the snippets, and was indeed presented with each snippet separately; thus our approach is defensible.

External validity refers to how these results generalize. Our data is limited to the Buse data, consisting of 12000 human judgements. The snippets were small, thus, we can say little about how these results might generalize to larger pieces of code or, what variables may be important when developers consider readability in various contexts. Further generalization requires more sources of readability ratings of snippets, source files, and patches.

Reliability refers to how replicable and how consistently the data was gathered and the study executed. Our study is easy to replicate but we do not know if the results will be stable on other datasets. Thus we rely on Buse *et al.*'s inter-rater reliability tests.

8. CONCLUSIONS

With the help of the human rankings that Buse *et al.* has graciously provided, we managed to demonstrate a very simple model of automatically rating code snippets as readable or not. Our model relies on two main measures: size and entropy. The greater the entropy of the snippet the more readable the snippet is, modulo the size. For a given entropy level, an increase in size contributes positively to readability. This result fundamentally makes sense because of the size limits of human short-term memory, the complexity derived from the information content of the underlying snippets, and the belief that basic typography affects readability.

We demonstrated that these simple, theoretically well-founded measures, used together, out-performed many of the models that Buse proposed. Our model of readability is easy to measure and quick to execute, thus it could be rapidly integrated into numerous version control products and source code monitoring tools. Finally, the simplicity of our models also potentially makes them easier for developers to use, should they seek to improve the readability of their code based on our models.

We acknowledge two important issues: first, Buse's readability ratings, although created via a well-organized human study, may not necessarily correspond to task-oriented reading in a particular context. Furthermore, these ratings were based on snippets of limited size, so their validity for large fragments is unclear. However we do believe that theoretically well-founded, parsimonious approaches to modeling of readability, as presented in this paper, will remain useful when more human-subject data on readability becomes available.

9. ACKNOWLEDGEMENTS

We extend our sincere thanks to Buse *et al.* for providing the original snippets and his tool for evaluating readability of Java source. All authors gratefully acknowledge support from the National Science Foundation, grant numbers 0964703 and 0613949. Devanbu gratefully acknowledges support from IBM Research and Microsoft Research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

10. REFERENCES

[1] The Zen of Python. http://www.python.org/dev/peps/pep-0020/. [Online; accessed 31-January-2011].

[2] K. Aggarwal, Y. Singh, and J. Chhabra. An integrated measure of software maintainability. In *Reliability and Maintainability Symposium, 2002. Proceedings. Annual*, pages 235–241. IEEE, 2002.

[3] R. M. Baecker and A. Marcus. *Human factors and typography for more readable programs.* ACM, New York, NY, USA, 1989.

[4] J. Börstler, M. Caspersen, and M. Nordström. *Beauty and the Beast: Toward a Measurement Framework for Example Program Quality*. Department of Computing Science, Umeå University, 2008.

[5] L. Briand and J. Wüst. Empirical studies of quality models in object-oriented systems. *Advances in Computers*, 56:97–166, 2002.

[6] R. Buse and W. Weimer. Learning a Metric for Code Readability. *Software Engineering, IEEE Transactions on*, 36(4):546–558, 2010.

[7] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp. Relating Identifier Naming Flaws and Code Quality: An Empirical Study. In *2009 16th Working Conference on Reverse Engineering*, pages 31–35. IEEE, 2009.

[8] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp. Exploring the influence of identifier names on code quality: An empirical study. In *14th European Conference on Software Maintenance and Reengineering*, March 2010. Pages 159-168.

[9] J. Cohen. *Applied multiple regression/correlation analysis for the behavioral sciences*. Lawrence Erlbaum, 2003.

[10] D. Coleman, D. Ash, B. Lowther, and P. Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, 2002.

[11] N. Coulter. Software science and cognitive psychology. *IEEE Transactions on Software Engineering*, pages 166–171, 1983.

[12] S. Dahiya, J. Chhabra, and S. Kumar. Use of genetic algorithm for software maintainability metrics' conditioning. In *Advanced Computing and Communications, 2007. ADCOM 2007. International Conference on*, pages 87–92. IEEE, 2008.

[13] F. Détienne and F. Bott. *Software design–cognitive aspects*. Springer Verlag, 2002.

[14] K. El Emam, S. Benlarbi, N. Goel, and S. Rai. The confounding effect of class size on the validity of object-oriented metrics. *Software Engineering, IEEE Transactions on*, 27(7):630–650, 2002.

[15] J. Elshoff and M. Marcotty. Improving computer program readability to aid modification. *Communications of the ACM*, 25(8):512–521, 1982.

[16] L. Etzkorn, S. Gholston, and W. Hughes Jr. A semantic entropy metric. *Journal of Software Maintenance and Evolution: Research and Practice*, 14(4):293–310, 2002.

[17] R. Flesch. A new readability yardstick. *Journal of applied psychology*, 32(3):221–233, 1948.

[18] R. Forax. Why extension methods are evil. http://weblogs.java.net/blog/forax/archive/2009/11/-28/why-extension-methods-are-evil. [Online; accessed 31-January-2011].

[19] B. Guzel. Top 15 best practices for writing super readable code. http://net.tutsplus.com/tutorials/html-css-techniques/top-15-best-practices-for-writing-super-readable-code/. [Online; accessed 31-January-2011].

[20] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[21] M. Halstead. *Elements of software science*. Elsevier New York, 1977.

[22] A. Hindle, M. Godfrey, and R. Holt. Reading beside the lines: Indentation as a proxy for complexity metric. In *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*, pages 133–142. IEEE, 2008.

[23] M. Kanat-Alexander. Readability and naming things. http://www.codesimplicity.com/post/readability-and-naming-things/. [Online; accessed 31-January-2011].

[24] J. Kearney, R. Sedlmeyer, W. Thompson, M. Gray, and M. Adler. Software complexity measurement. *Communications of the ACM*, 29(11):1044–1050, 1986.

[25] D. Kozlov, J. Koskinen, M. Sakkinen, and J. Markkula. Assessing maintainability change over multiple software releases. *Journal of Software Maintenance and Evolution: Research and Practice*, 20(1):31–58, 2008.

[26] J. Kumar Chhabra, K. Aggarwal, and Y. Singh. Code and data spatial complexity: two important software understandability measures. *Information and software Technology*, 45(8):539–546, 2003.

[27] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transactions on Software Engineering*, 34(4):485, 2008.

[28] J. Lin and K. Wu. A Model for Measuring Software Understandability. In *Computer and Information Technology, 2006. CIT'06. The Sixth IEEE International Conference on*, page 192. IEEE, 2006.

[29] J. Lin and K. Wu. Evaluation of software understandability based on fuzzy matrix. In *Fuzzy Systems, 2008. FUZZ-IEEE 2008.(IEEE World Congress on Computational Intelligence). IEEE International Conference on*, pages 887–892. IEEE, 2008.

[30] A. Mohan, N. Gold, and P. Layzell. An initial approach to assessing program comprehensibility using spatial complexity, number of concepts and typographical style. In *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, pages 246–255. IEEE, 2005.

[31] N. Naeem, M. Batchelder, and L. Hendren. Metrics for measuring the effectiveness of decompilers and obfuscators. In *Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on*, pages 253–258. IEEE, 2007.

[32] P. Peduzzi, J. Concato, E. Kemper, T. Holford, and A. Feinstein. A simulation study of the number of events per variable in logistic regression analysis* 1. *Journal of clinical epidemiology*, 49(12):1373–1379, 1996.

[33] D. Raymond. Reading source code. In *Proceedings of the 1991 conference of the Centre for Advanced Studies on Collaborative research*, pages 3–16. IBM Press, 1991.

[34] P. Relf. Tool assisted identifier naming for improved software readability: an empirical study. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, page 10. IEEE, 2005.