An Approach Towards Merging Grammars

Isaac Griffith and Rosetta Roberts
Empirical Software Engineering Laboratory
Informatics and Computer Science
Idaho State University
Pocatello, Idaho, 83208
Email: {grifisaa@isu.edu, roberose@isu.edu}

***Abstract*—Multilingual parsing has been found to be a difficult task which island grammar based parsers have been demonstrated to excel at. In order to automate the process of creating multilingual island grammar based parsers, which has not been done, there is a need for a method of automatically merging grammars. Such a method is presented in this paper. This paper also evaluates the change in overall maintenance effort and complexity of grammars caused by varying the threshold value used to determine when similar productions can be merged. To show this, we conducted two experiments each using a factorial design. Data for both experiments was gathered from the ANTLR grammars-v4 repository. Both experiments show that any merging of similar productions decreases the maintenance effort and complexity of the resulting grammar. The decrease was found to be more significant for lower threshold values. This paper presents the initial steps towards the automated construction of island and tolerant grammars for the purpose of multilingual parsing.**

***Index Terms*—Island Grammars, Automated Grammar Formation, Software Language Engineering**

I. Introduction

Modern software development practice has led to an increasing number of software systems created using multiple languages. As an example, the modern web application is typically constructed using 5 or more languages (e.g. SQL, Java, TypeScript, HTML, CSS). Such multilingual codebases present a difficult challenge to the development and maintenance of source code analysis tools [1]. Current source analysis tools typically address this challenge using a combination of multiple parsers (one per supported language).

Code analysis tools have become an essential part of modern coding [2]. Using these tools, software engineers find bugs, identify security flaws, increase product quality, and comply with business rules and regulations. These tools have been integrated into software build processes and integrated into their pipelines for quality assurance and other services (e.g. SonarQube™[1]).

Island grammars have been shown to be a solution to the problem of developing multilingual parsers [3]. Though useful, the technique requires the manual combination of selected components from source grammars, a process which can be cumbersome to maintain as these grammars evolve. To overcome this, we propose an automated method to reduce the initial time-consuming manual process and the further difficulty of maintaining the constructed island grammar. In this paper we discuss the capability to correctly combine grammar productions together without reducing the grammar's ability to define key aspects of interest, a key component to the automated construction of island grammars.

The automated merging of grammars is an important and necessary step in the evolution of island grammar research. The capability to automate grammar merging addresses the key issues found in the initial construction and further maintenance of island grammars. To evaluate this hypothesis, we used the Goal-Question-Metric (GQM) paradigm [4] to form the following research goal: Evaluate an automated approach for the purpose of automating the merging of grammar rules with respect to the maintenance effort and complexity from the point of view of software language engineers in the context of the creation of tolerant and island grammars. The solution proposed will allow tool designers to develop source code analysis tools which support multiple programming languages in an efficient and maintainable way.

Organization

The remainder of this paper is organized as follows. Section II discusses the theoretical foundations and alternative approaches related to this work. Section III details our approach to automate the merging of grammars which forms the foundation of an initial approach to automating the creation of multilingual parsers, which we call SIGMA. Section IV details the design of experiments which evaluate the proposed approach and its parameterization. Section V presents the results of the experiments. Section VI presents a discussion of the results and their interpretation in light of other studies. Section VII details the threats to the validity of this study. Finally, this paper is concluded in Section VIII.

II. Background and Related Work

A. Theoretical Foundations

A context free grammar, G, can be described as $G = (V, \Sigma, P, S)$ [5], where V is the set of non-terminal symbols, Σ is the set of terminal symbols, $P \subseteq V \times (V \cup \Sigma)^*$ is the set of productions describing how the symbols of V can be substituted for other symbols, and $S \in V$ is the starting symbol. Each production is written as $a \rightarrow b$ with $a \in V$ and $b \in (V \cup \Sigma)^*$. When b is the empty string, the production is denoted by $a \rightarrow \varepsilon$. A string, s, is called a *valid sentence* for

[1] https://sonarqube.org

a grammar if it can be created by repeated application of the productions of that grammar [6]. $L(G)$ denotes the set of all valid sentences, or language, of grammar G.

Island Grammars are a specialized form of context free grammar which include the addition of a set of interests. These interests are used to focus the grammar on the set of language components most of interest to the grammar developers. An island grammar is formally defined as the following tuple $G' = (V, \Sigma, P, S, I)$ [6], where I is the set of interests. These interests, as noted, define the components to which the grammar is focused, these are also known as islands. The remaining components of a language are reduced down into one or more catchall productions referred to as water [6]. This has the effect of reducing the complexity of the grammar. An island grammar, G', derived from a grammar, G, has a language $L(G')$ which satisfies the following property: $L(G') \supset L(G)$. Island grammars offer several advantages over regular grammars for many applications including faster development time, lower complexity, and better error tolerance. Due to this, island grammars have been used for many applications including documentation extraction and processing [7], impact analysis [8], and extracting code embedded in natural language documents [9], [10]. Of particular interest to our research is their use for creating multilingual parsers [3], which inspired this research, and research into the development of tolerant grammars [11]–[13].

B. Alternative Approaches

Using an intermediate representation (IR), such as Java™ bytecode or Microsoft IL, is a common method for performing operations on multilingual code bases. This allows code analysis tools to be built around the language(s) defined by the IR rather than having to deal with each of the different languages individually. However, using an IR doesn't work when analyzing systems created from languages that do not share a common IR [14]. Using an IR also doesn't work if performing manipulations of source code or if there isn't a clear correspondence between a source language and its IR. An example of this can be seen for PIT [15], a Java mutation testing framework. Even though their system is designed around transforming and manipulating Java bytecode, the system struggles with handling bytecode generated by other JVM languages. However, they still have had limited success with other JVM languages [16].

Another approach that's been developed for analyzing multilingual systems is through the use of a separate parser for each language. Constructing and integrating each parser for each language can be both difficult and time consuming, and thus it is better to use existing parsers [17]. Using existing parsers, one can modify the abstract syntax trees into a language independent form. The problem one contends with in this approach is these parsers may be written in a variety of languages. The use of an existing Island Grammar [3] may alleviate these issues, but typically they tend to be specific to their particular problem and their construction is currently a time-consuming manual process.

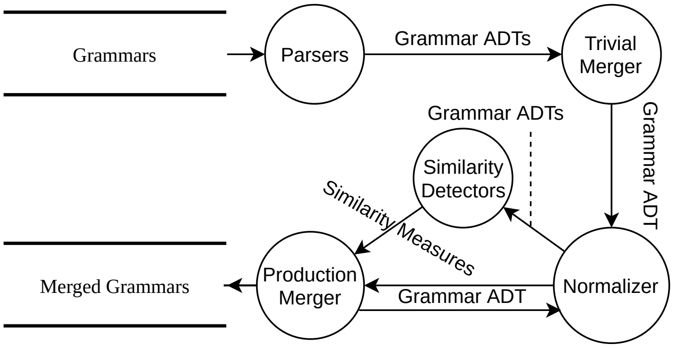

Fig. 1. Data flow diagram of merging process.

$\langle S \rangle$ ::= <A> | <B>
$\langle Y \rangle$ ::= <A> | 'y'
$\langle A \rangle$ ::= 'a' ε <B> <C>
$\langle C \rangle$ ::= 'c'
$\langle B \rangle$ ::= 'b' 'd'

(a) Grammar G_1.

$\langle S \rangle$ ::= <C> | <D>
$\langle Z \rangle$ ::= <S> | 'z'
$\langle C \rangle$ ::= 'c'
$\langle D \rangle$ ::= 'a' 'd' ('e' | 'c')
| (<C> | 'b')

(b) Grammar G_2.

Fig. 2. Example grammars used to demonstrate merging procedure.

III. Approach

In order to generate an island grammar from one or more source grammars, there is a need to merge components of the source grammar(s) to reduce or deduplicate the final grammar. Such a technique can be used to reduce the island components. Thus, the focus of this paper is the development of a technique to merge grammars together, and is depicted in Fig. 1. The process is as follows:

1. Parse the input grammars.
2. Trivially merge the grammars into a single grammar.
3. Normalize the merged grammar.
4. Calculate the similarity between each pair of rules.
5. Merge the two most similar rules.
6. Repeat steps 4 and 5 until the two most similar rules are no longer similar.
7. Final Normalization of Merged Grammar
8. Output the merged grammar.

The remainder of this section presents the details of each of these steps and utilizes a running example based on the the two grammars depicted in Fig. 2a and Fig. 2b.

A. Parsing

The initial step of the process is the translation of the textual representations of the grammars into an instance of our domain model, depicted in Fig. 3. In our experiments we analyzed grammars stored in the ANTLR 4 grammar format. Thus, to create instances of our model, we processed the grammars using an ANTLR 4 grammar parser. This parser constructs an abstract syntax tree which is then converted into an instance of the domain model.

The conversion process is complicated by the fact that the ANTLR 4 representation is not fully compatible with

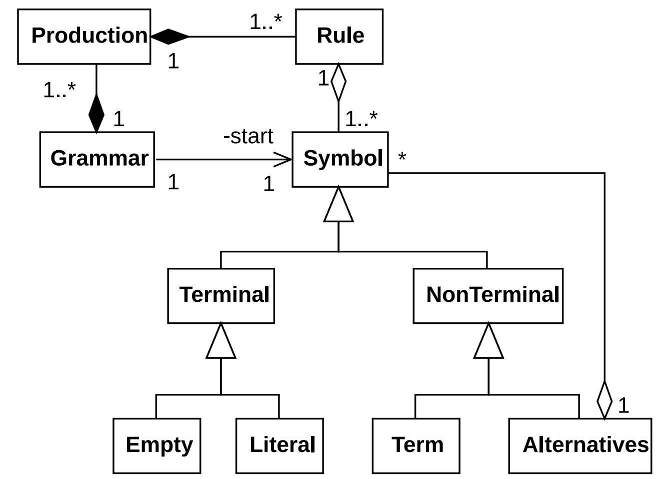

Fig. 3. Class diagram of internal domain model used to represent grammars throughout the merging process.

our BNF based domain model. To resolve this, we removed or converted all non-BNF features except for *dot*, *character range*, *character class*, and *not* rules. These operators act as they do in any typical regular expression language. The *dot*, *character range*, and *character class* operators, were converted to special terminal symbols. *Not* rules were retained by extracting each into its own rule. For example:

```
A: 'a' . ~('b' | 'c')
```

would be converted to:

$\langle A \rangle$ ::= ‘a’ DOT $\langle$*Generated-1*$\rangle$
$\langle$*Generated-1*$\rangle$ ::= ~$\langle$*Generated-2*$\rangle$
$\langle$*Generated-2*$\rangle$ ::= ‘b’ | ‘c’

The *not* rules are ignored completely by this process except for when detecting and merging duplicate rules.

B. Trivial Merge

The initial merge is the combination of the start productions of the grammars to be combined. First, a new start symbol is created in the new grammar. Productions are also added from this symbol to each of the source grammars' start productions. The trivial merging of grammars G_1 and G_2 yields the grammar G_3 depicted in Fig. 4. Where the original source grammar start productions are renamed S1 and S2 representing the start productions for both G_1 and G_2, respectively.

C. Normalization

Next, the trivially merged grammar is normalized. The normalization process works to eliminate unused rules, simplify productions, merge equivalent rules, eliminate unit rules, expand productions, and collapse compatible productions. We leave the details of this process to another paper because the complexity of this process can't be fully explained in this paper without detracting from this paper's focus on the merging of similar rules.

$\langle S \rangle$ ::= $\langle S_1 \rangle$ | $\langle S_2 \rangle$
$\langle S_1 \rangle$ ::= $\langle A \rangle$ | $\langle B \rangle$
$\langle Y \rangle$ ::= $\langle A \rangle$ | ‘y’
$\langle A \rangle$ ::= ‘a’ ε $\langle B \rangle$ $\langle C_1 \rangle$
$\langle C_1 \rangle$ ::= ‘c’
$\langle B \rangle$ ::= ‘b’ ‘d’
$\langle S_2 \rangle$ ::= $\langle C_2 \rangle$ | $\langle D \rangle$
$\langle Z \rangle$ ::= $\langle S \rangle$ | ‘z’
$\langle C_2 \rangle$ ::= ‘c’
$\langle D \rangle$ ::= ‘a’ ‘d’ (‘e’ | ‘c’)
| ($\langle C_2 \rangle$ | ‘b’)

Fig. 4. Grammar G_3. The result of trivially merging G_1 and G_2.

D. Measuring Rule Similarity

The fourth step of this approach measures the similarity between each pair of productions. This measure is used by the fifth step to identify merge candidate pairs. In this section, describe the two similarity metrics corresponding to the two forms (defined in Section III-C) of productions produced by the normalization process. We note productions of different forms will have a similarity of 0.

1) Term Similarity: The first similarity metric measures the similarity between two productions of form 1. As an example, assume we have the following two productions:

$\langle P_a \rangle$ ::= ‘a’ $\langle A \rangle$ ‘b’ ‘c’
$\langle P_b \rangle$ ::= $\langle A \rangle$ $\langle B \rangle$ ‘b’ ‘a’

The first step towards measuring the similarity between these productions is aligning the productions using the Longest Common Subsequence (LCS) algorithm [18].

```
'a' <A>     'b'     'c'
    <A> <B> 'b' 'a'
```

Once aligned, the number of aligned terms is divided by the total number of terms, as described in (1):

$$S_1 = \frac{2|\mathrm{LCS}(P_a, P_b)|}{|P_a| + |P_b|}. \qquad (1)$$

Applying (1) to the example, `<A>` and `'b'` are each counted twice because they occur in both sequences of terms. The total number of terms across both productions is 8. The total similarity score is computed as, $S_1 = \frac{4}{8} = .5$.

2) Alternatives Similarity: The second similarity metric measures the similarity between two productions of form 2. As an example, assume we have the following two productions:

$\langle P_a \rangle$ ::= ‘a’ | $\langle A \rangle$ | ‘b’ | ‘c’
$\langle P_b \rangle$ ::= $\langle A \rangle$ | $\langle B \rangle$ | ‘b’ | ‘a’

Similar to the previous metric, this metric calculates the number of common alternatives divided by the total number of alternatives. However, because the order of the alternatives does not matter, a different approach to measure the common alternatives is applied. In this approach an alternative is counted as common to both if it occurs in both productions. Equation (2) describes how to calculate the similarity score using this method:

$$S_2 = \frac{2|P_a \cap P_b|}{|P_a| + |P_b|}. \quad (2)$$

Applying (2) to the example, the common elements are `<A>`, `'a'`, and `'b'`. The similarity score is computed as $\frac{2 \cdot 3}{8} = 0.75$.

E. Merging Similar Rules

Once the two most similar productions are detected they are then merged together. Similar to the process of measuring similarity, there are two form specific processes for merging similar productions. This process relies upon a minimal similarity threshold, M_s, above which similar productions are merged.

1) Merging Similar Terms: To merge two productions of the first form, the LCS alignment produced while measuring the similarity between terms is used. Initially, each pair of subsequences that do not align are identified. In the previous example, the identified unaligned pairs of subsequences are: (`'a'`, ε), (ε, `<B>`), and (`'a'`, `'c'`). Each pair of unaligned subsequence is then substiuted with the union of the two subsequences. As an example, the prior example sequences of terms would merge to: $\langle P_{a+b} \rangle$ ::= (`'a'` | ε) $\langle A \rangle$ (ε | $\langle B \rangle$) `'b'` (`'c'` | `'a'`).

2) Merging Similar Alternatives: Merging two productions of the second form is simpler than the first form. The merged production produced simply contains all alternatives that are in either constituent production. As an example, the prior example productions $\langle P_a \rangle$ and $\langle P_b \rangle$ would be merged to the following: $\langle P_{a+b} \rangle$ ::= `'a'` | $\langle A \rangle$ | $\langle B \rangle$ | `'b'` | `'c'`.

F. Grammar Output

At the end of this process, we output the merged and transformed grammar. The output is generated by simply visiting every production in the grammar. As each production is visited a corresponding ANTLR 4 production is generated and written out.

IV. Experimental Design

This section describes the overall experimental design used to evaluate the grammar merging approach presented within this paper. The following subsections detail the separate aspects of our experimental design.

A. Goals, Hypotheses, and Variables

This subsection describes the refinement of our initial research goal, defined in Section I, into a set of actionable research questions and metrics. Based on this set of research questions we also identified the variables used in statistical models driving our analytical procedures. We begin with the research questions and metrics.

Following the GQM paradigm, research goal RG can be decomposed into the following set of research questions:

RQ1 What is the effect that this process has on the maintenance effort between the source grammars and the grammar produced by this approach?
Rationale: *It is expected that the merging of grammar components will reduce the maintenance effort required.*

RQ2 What is the effect that this process has on the complexity between the source grammars and the grammar produced by this approach?
Rationale: *It is expected that the merging and reduction of grammar components will reduce the complexity of the grammar, thus making the grammar easier to understand and read.*

In addition to these research questions we have selected the following metrics to assess the results of the approach used:

M1 Effort – To assess the effort required to maintain a grammar, we utilize the Halstead Effort measure for grammars as defined by Power and Malloy [19].

M2 Complexity – To assess the complexity of a grammar, we utilize McCabe's Cyclomatic Complexity metric for grammars defined by Power and Malloy [19].

The dependent variables in the experiments, as indicated by the above research questions, are maintenance effort and complexity. Specifically, we are concerned with the change between the trivial merge state and the final grammar in terms of the effort and complexity of the grammar. Thus, the dependent variables of concern are:

- ΔHAL – the change in Halstead Effort as measured after the normalization phase prior to the final merge phase, and after the final merge phase and before generating the output grammar.
- ΔMCC – the change in complexity as measured after the normalization phase prior to the final merge phase, and after the final merge phase and before generating the output grammar.

The independent variables we are concerned with are:

- Similarity Threshold – the parameter guiding the similarity measurements used in the merging process. The values used in the experiments are 0.001, 0.25, 0.5, 0.75, and 1.0.
- Size – the size of the grammar as defined by measuring its number of productions (PROD) [19], and threshold this value into three distinct categories: Small, Medium, and Large, as defined in Section IV-C.

B. Design

To evaluate the approach we elected to conduct two experiments. The first experiment evaluates the effect the approach has on the maintenance effort necessary to maintain a merged grammar as compared to its combined source grammars. The second experiment evaluates the effect the approach has on the complexity of the merged grammar as compared to its combined source grammars. These experiments utilize a Factorial Design, with a single dependent variable (ΔHAL, ΔMCC), a treatment factor $SimilarityThreshold$ and the grouping factor $Size$.

TABLE I
GRAMMARS RANDOMLY SELECTED FROM EACH SIZE CATEGORY USED IN THE EXPERIMENTS.

Category	Grammars
S	brainfuck, cmake, csv, inf, lcc, pdn,
	properties, quakemap, sexpression, tsv, url, useragent
M	cto, dart2, flatbuffers, fusion-tables, lua, pascal
	python2, romannumerals, sgf, stacktrace, webidl, z-ops
L	cql3, edif300, fortran77, idl, informix, java9
	kotlin, objc-two-step, powerbuilder, rexx, sharc, swift2

C. Experimental Units

In these experiments, the experimental units are pairs of grammars selected from the ANTLR 4 [20] grammar repository[2]. The sys-verilog grammar was excluded because of errors while parsing it. At the time of this writing, the repository contained 198 individual grammars from a variety of general purpose and domain specific languages.

The process used to select the grammar pairs for each experiment is depicted in Fig. 5 and works as follows. Initially, for each of the grammars in the repository we collected a combination of metadata and metric measurements. The metadata consists of the language represented by the grammar, the version of that language (if applicable) and the following metrics (selected from the metrics suite by Power and Malloy [19]):

- TERM – the number of terminals.
- VAR – the number of defined non-terminals.
- PROD – the number of productions.
- MCC – McCabe's cyclomatic complexity.

Using the resulting measures of these metrics, the grammar dataset was subdivided into three categories (Small, Medium and Large) based on the logarithm of PROD (as the values were log-normal distributed). This subdivision was based on statistically construction thresholds, as per Lanza and Marinescu [21]. Category threshold values are defined as: Small-Medium: $10^{\mu_{PROD}-\sigma_{PROD}} = 10^{1.8404-0.5371} = 20.1084$ and Medium-High: $10^{\mu_{PROD}+\sigma_{PROD}} = 10^{1.8404+0.5371} = 238.4995$. Using these thresholds each grammar is then grouped into one of the three categories.

Using these categories as the grouping factor in the experiments, we can then begin the sampling process. As we have selected a 3×5 factorial design, each replication of each of the two experiments requires 15 grammar pairs (5 per size category). Based on our replication analysis (described in Section IV-E) we have identified a need for a total of 5 replications. Thus, for each size category we require a total of 50 grammar pairs per experiment yielding a need for 100 total grammar pairs. To meet this requirement we randomly select (without replication) 12 grammars, per size category. From these 12 grammars there are $\binom{12}{2} = 66$ combinations without replication of which we randomly select 5 pairs per

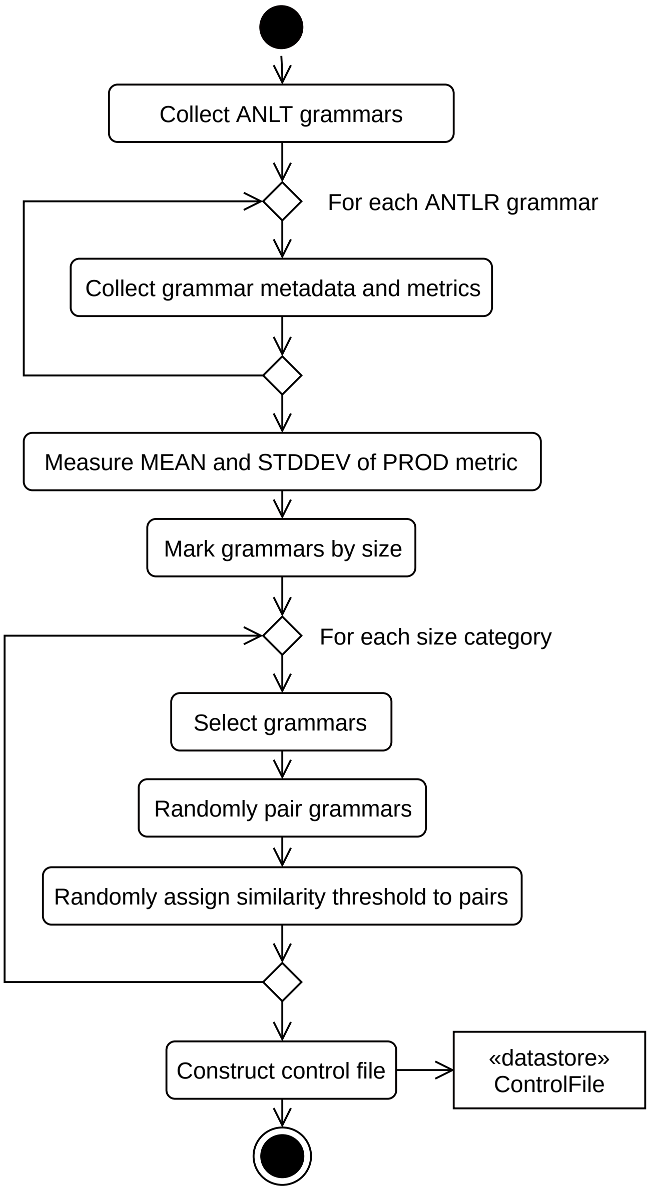

Fig. 5. Activity diagram describing how control files representing each replication are created.

[2] https://github.com/antlr/grammars-v4

replication per experiment. The grammars selected and their pairings per experiment/replication are depicted in Table I.

Once the grammar pairs are selected, they are assigned a treatment value for use during the experiment execution and data collection phase. For each size category per replication per experiment, the set of five grammar-pairs are assigned, at random, a level of the similarity threshold treatment. The possible levels that can be assigned are 0.001, 0.25, 0.5, 0.75, and 1.0 where 1.0 is the control level.

Finally, the data generated as part of the selection process is used to construct an experiment control file. This file is used to direct the experimental execution system and to ensure the validity of the process. Each control file is simply an ordered set of triples where each triple consists of the following information:

- Grammar Pair - the pair of grammars to be merged together, as selected during the selection phase.
- Treatment - the similarity threshold value assigned to the grammar pair during the selection phase.
- Size - value of the size category, for use in constructing the data table.

Each replication has a separate control file to control its execution. Thus, for each replication, the control file triples are created and put into a list which is then randomized. This randomized list is then output to a file which is later read in by the experimental execution system.

D. Data Collection Procedures

Data collection is performed via the experiment execution system specifically developed for this study. This system is controlled via the control files created during the experimental unit selection process. The data collection process follows the activity diagram depicted in Fig. 6.

Initially, the control file is read into the system. Then, for each of the triples in the list, the following occurs: 1.) The selected similarity threshold value is applied. 2.) The grammars located, read in, and trivially merged together. 3.) The combined grammar's effort or complexity is then measured and recorded. 4.) The final merging process is applied resulting in the final grammar. 5.) The resulting grammar's effort or complexity is then measured and recorded.

Once all experimental units have been processed the results are exported to a data table. This process is repeated for each replication of each experiment. The results from each replication of each experiment are then combined into a single data table. The combined data table is then used during the analysis phase.

E. Analysis Procedures

As described in Section IV, both experiments utilize a Factorial design [22]. Typically, a factorial design experiment will utilize ANOVA to determine if there is a difference between the effects of the factors. In this case, the statistical model used is as follows:

$$y_{ijk} = \mu + st_i + size_j + (st * size)_{ij}\epsilon_{ijk}$$

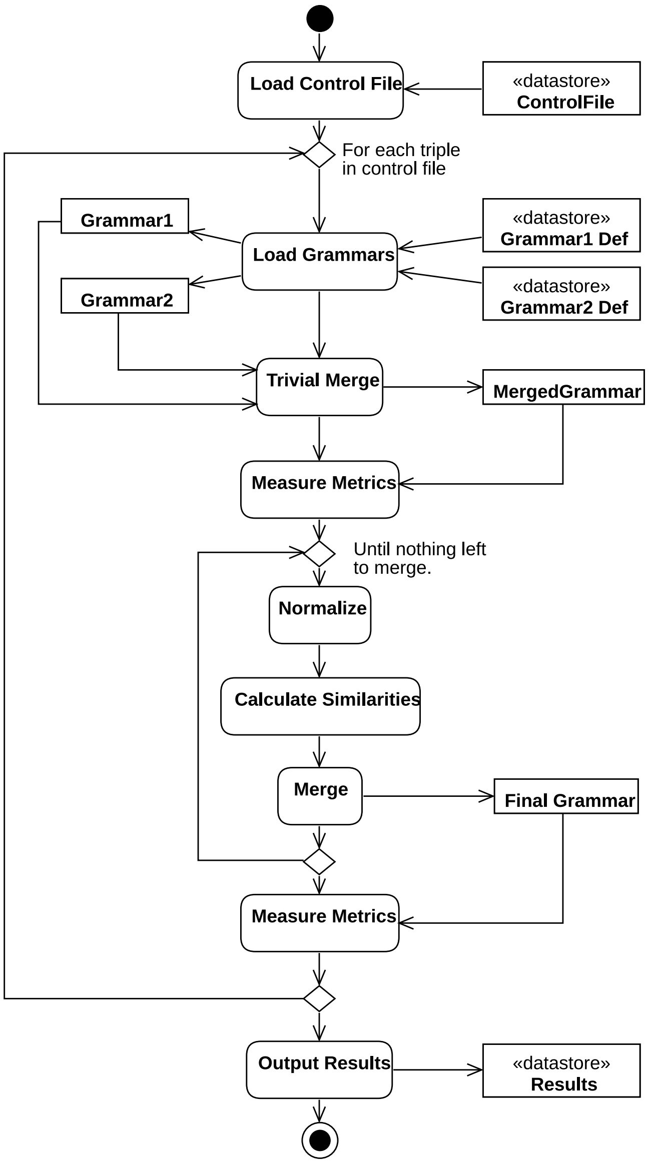

Fig. 6. Activity diagram showing locations in the merging process where metrics are gathered.

Where:

- y_{ijk} is the kth value of the observation (either ΔMCC or ΔHAL) associated with the ith similarity threshold level and jth size level.
- μ is the baseline mean
- st_i is the ith level of similarity threshold effect.
- $size_j$ is the jth level of size effect.
- $(st * size)\_ij$ is the similarity threshold * size interaction effect.
- ϵ_{ijk} is the random error of the kth observation from the (i, j)th cell.

The hypotheses to be tested in this case are as follows:

- $H_{1,0}$: The effect of the levels of the interaction term are equal.

- $H_{1,A}$: There is at least one difference between interaction level effects.
- $H_{2,0}$: The effects of the levels of similarity threshold are equal.
- $H_{2,A}$: There is at least one difference between similarity threshold level effects.
- $H_{3,0}$: The effects of the levels of size are equal.
- $H_{3,A}$: There is at least one difference between size level effects.

ANOVA has several assumptions that must first be verified. First, is the assumption of homogeneity of variance. To evaluate this assumption, we will use Levene's Test [23]. The next assumption is that both the factors and errors are normally distributed. To verify this assumption, we will use the Anderson-Darling Test for Goodness of Fit [24] to the normal distribution for both the errors and factor effects. Finally, there is the assumption of independence of the observations, which is valid due to the nature of the process.

In the case of any violations of these assumptions we will attempt to correct the violation using a Box Cox transformation. In the case that the assumptions are violated beyond the capability to correct, we will be forced to utilize a non-parametric approach. Specifically, we will use a permutation F-test [25], in place of ANOVA. The permutation F-test also has a set of assumptions that must be met prior to use. The model for the permutation F-test is the same as that of ANOVA.

In either case, the next step is to conduct the hypothesis test and make a decision. In this case we have selected an α threshold of 0.95. In the case that we reject $H_{1,0}$ we will then conduct a multiple-comparison procedure to compare the individual effects of each level of the similarity threshold factor. In these experiments the similarity threshold level of 1.0 to be the control (as noted in Section IV-B). Dunnett's [26] multiple comparison procedure will be selected if ANOVA is used, and Steel's [27] multiple comparison procedure (a non-parametric technique analogous to Dunnett's) if using the permutation F-test. Both of these comparison procedures control the error for multiple comparisons and allow the ability to compare against a control. In the case of Dunnett's Test we will be evaluating the following hypotheses:

- $H_{4,0}$: There is no difference between the mean effects of similarity threshold effects and control effect.
- $H_{4,A}$: There is a difference between at least one similarity threshold level and control.

In the case of Steel's Test we will be evaluating the following hypotheses:

- $H_{4,0}$: There is no difference between the median effects of similarity threshold effects and control effect.
- $H_{4,A}$: There is a difference between at least one similarity threshold level and control.

For either of these tests we will use an α threshold of 0.95.

Additionally, we are interested if there is a strict order of the effect on ΔHAL or ΔMCC for the levels of the similarity threshold factor. To evaluate this we have selected to utilize the Jonhckheer's trend test. This is a non-parametric test to determine if there is an *a priori* ordering within independent samples [28]. The hypotheses to be tested are as follows:

- $H_{5,0}$: There is no difference in median effects of the similarity threshold levels.
- $H_{5,A}$: There is a increasing trend in the effects of the similarity threshold levels, with at least one being a strict inequality.

Finally, we will also conduct a sample size analysis to determine the number of repetitions necessary to achieve the power level necessary for the experiments.

TABLE II
DESCRIPTIVE STATISTICS.

Size Category	**Mean ΔHAL**	**Mean ΔMCC**
Small	1290.2	32.32
Medium	18573	112.7
Large	149854	842.4

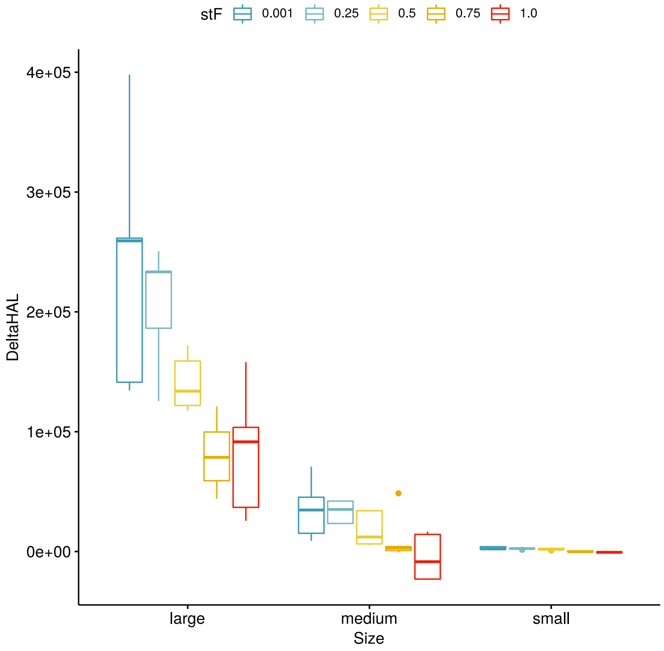

Fig. 7. Results of running the Delta HAL experiment in the form of boxplots.

V. Results

A. Descriptive Statistics

This subsection describes the data collected via a series of descriptive statistics. Table II shows the mean values of ΔHAL and ΔMCC for each value of the size category in the respective experiments. Along with this table the dispersion of the both ΔHAL and ΔMCC across size and similarity threshold values is displayed in Fig. 7 and Fig. 8, respectively.

As we can see from the Residuals vs. Fitted plot, depicted in Fig. 9, the homogeneity of variance assumption is not met. To validate this observation we used Levene's Test. In this test, the null-hypotheses (H_0) asserts that the variance

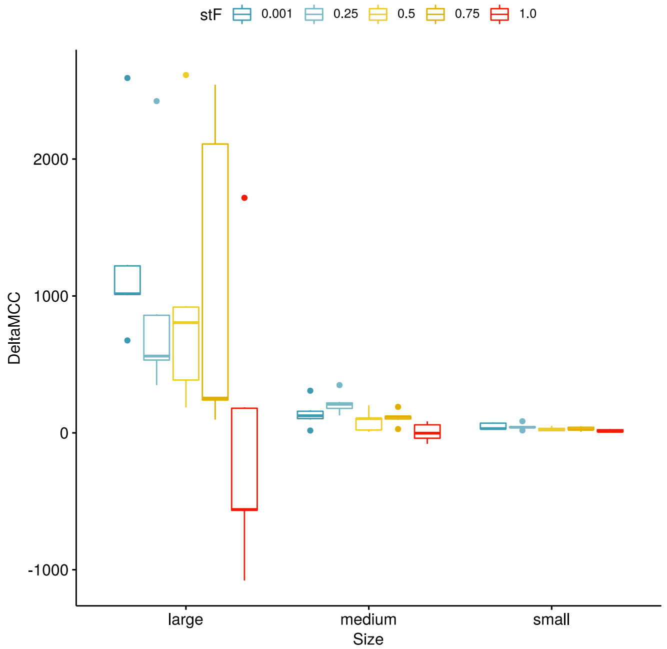

Fig. 8. Results of running the Delta MCC experiment in the form of boxplots.

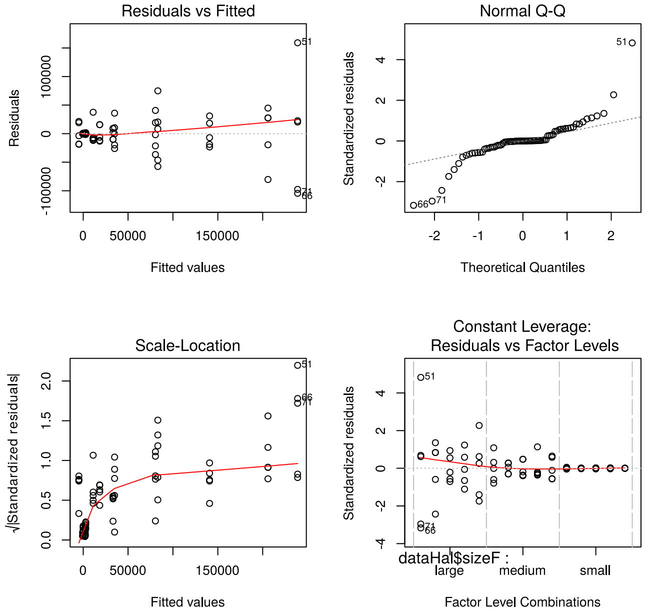

Fig. 9. Delta Hal experiment Q-Q plots.

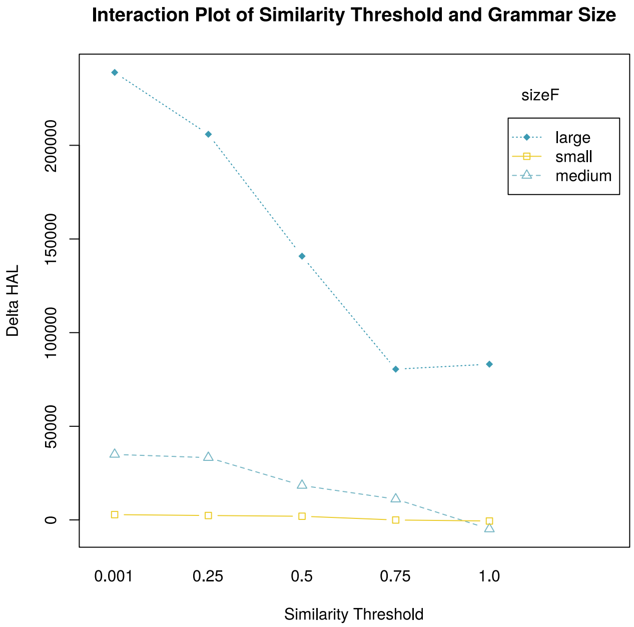

Fig. 10. Delta HAL experiment interaction plot.

is equally distributed. As expected, from our observations of the Residuals vs Fitted Values plot, the test produced an F-value of 19.962 and a p-value of 1.267e-07 indicating that we should reject the null hypothesis that there is a homogeneity of variance.

The Normal Q-Q plot, depicted in Fig. 9, indicates that the normality of the residuals assumption is also violated. To validate this observation we conducted both an Anderson-Darling GOF test against the Normal distribution. In this test the null-hypothesis (H_0) is that the empirical distribution is Normal. The results show that the A statistic has a value of 6.8257 and an associated p-value of $<$ 2.2e-16 which indicates that we can reject the null-hypothesis. These results confirm our initial observation that the results are not normally distributed.

The interaction plot, depicted in Fig. 10, indicates that there is an interaction between the small and medium levels as the similarity threshold changes from 0.75 to 1.0. This indicates that the assumption that the block and treatment do not interact and that an RCBD design is inappropriate and a factorial design would be more accurate.

As we can see from the Residuals vs. Fitted plot, depicted in Fig. 11, the homogeneity of variance assumption is not met. To validate this observation we used Levene's Test. In this test, the null-hypotheses (H_0) asserts that the variance is equally distributed. As expected, from our observations of the Residuals vs Fitted Values plot, the test produced an F-value of 29.856 and a p-value of 3.61e-10 indicating that we should reject the null hypothesis that there is a homogeneity of variance.

The Normal Q-Q plot, depicted in Fig. 11, indicates that

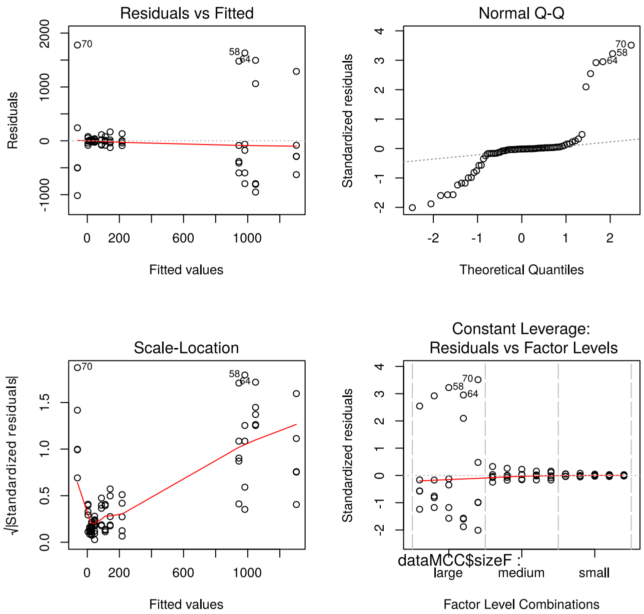

Fig. 11. Delta MCC Experiment Q-Q plots.

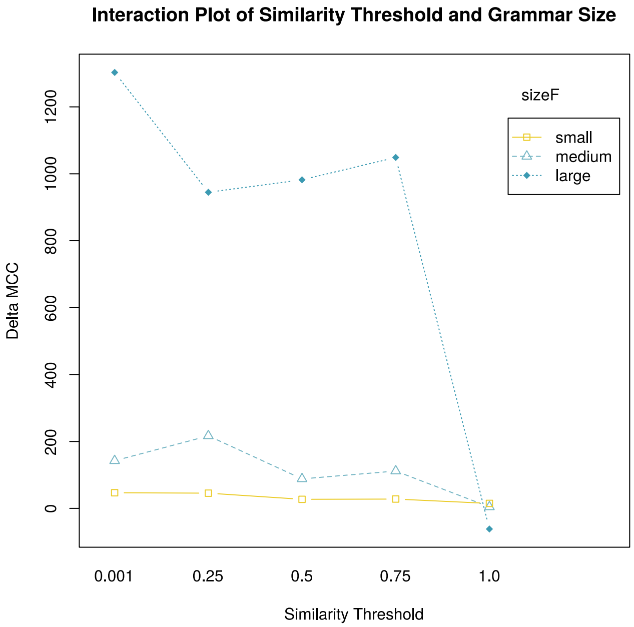

Fig. 12. Delta MCC experiment interaction plot.

the normality of the residuals assumption is also violated. To validate this observation we conducted both an Anderson-Darling GOF test against the Normal distribution. In this test the null-hypothesis (H_0) is that the empirical distribution is Normal. The results show the A statistic has a value of 9.9558 and an associated p-value of < 2.2e-16 which indicates that we can reject the null-hypothesis. These results confirm our initial observation that the results are not normally distributed.

The interaction plot, depicted in Fig. 12, indicates that there is an interaction between the small and medium levels as the similarity threshold changes from 0.75 to 1.0. This indicates that the assumption that the block and treatment do not interact and that an RCBD design is inappropriate and a factorial design would be more accurate.

B. Hypothesis Testing

This subsection details the results of the ΔHAL experiment and ΔMCC experiment. In both cases we initially executed a sample size analysis to determine the number of replications to execute to achieve a power level of .95. The results of this analysis indicate the need to execute a total of 5 replications of each experiment.

1) ΔHAL Experiment: We conducted an experiment using a factorial design based on the levels of the factors size and similarity threshold. Our goal was to determine if the algorithm when applied had an effect on the change in Halstead Effort for the combined grammars, as measured by ΔHAL. Because of the violations to the assumptions of ANOVA, we opted to utilize non-parametric methods in our analysis. The initial analysis utilizes a permutation F-test approach to evaluate if there is any difference among factor levels. The results was had a p-value of $< 2.2 \times 10^{-16}$ indicating that we could reject $H_{1,0}$. We also note that the interaction indicated in Fig. 10 was shown to be significant with a p-value of $< 2.2 \times 10^{-16}$.

We were interested in determining if there was any difference between the control level (1.0) and other levels of the similarity threshold. To evaluate this we used Steel's multiple comparison against a control. The results of this test indicated for each level that there was a significant difference and that each level the mean ΔHAL was lower than that of the control value.

Finally, we were interested in evaluating the selected levels of the similarity threshold (excluding control) and if there was an implied order (that is there is a decreasing order to the change in ΔHAL) as the level value is lowered in the similarity threshold. To evaluate this we used Jonchkheere's trend test for ordered differences among classes, which yields a JT statistic value of 767 and a p-value of 4×10^{-4}, when executed using 10000 permutations. This indicates that we can reject $H_{3,0}$ that there is no *a priori* ordering among levels of similarity threshold on median ΔHAL.

2) ΔMCC Experiment: We conducted an experiment using a factorial design based on the levels of the factors size and similarity threshold. Our goal was to determine if the algorithm when applied had an effect on the change in Complexity for the combined grammars, as measured by ΔMCC. Because of the violations to the assumptions of ANOVA, we opted to utilize non-parametric methods in our analysis. The initial analysis utilizes a permutation F-test approach to evaluate if there is any difference among factor levels. The results was a p-value of 0.1429 indicating that we failed to reject $H_{1,0}$.

We tested if there was any difference between the control level (1.0) and other levels of the similarity threshold. To evaluate this, we used Steel's multiple comparison against a control. The results of this test indicated for each level that there was a significant difference and that for each level the mean ΔMCC was lower than that of the control value.

Finally, we evaluated the selected levels of the similarity threshold (excluding control) to if there was a decreasing order to the change in ΔMCC as the similarity threshold values is lowered. To do this, we used Jonchkheere's trend test for ordered differences among classes, which yielded a JT statistic value of 742 and a p-value of 1×10^{-4} when executed using 10000 permutations. This indicated that we can reject $H_{3,0}$: that there is no *a priori* ordering among levels of similarity threshold on median ΔMCC.

VI. Interpretation

The results indicate that the proposed merge process embodied in our algorithm reduces the Halstead Effort and McCabe Cyclomatic Complexity of a combined grammar at each threshold below the control threshold, regardless of size. The results also indicate there is a increasing order to the amount of change in Halstead Effort and cyclomatic complexity caused by the algorithm as the similarity threshold decreases. This implies that smaller values of the similarity threshold produce better results.

Since this was a randomized experiment, one may infer that the difference in Halstead Effort and McCabe Cyclomatic Complexity was caused by the difference in similarity threshold. Because the subjects were selected randomly from the population of ANTLR grammars, we can extend this inference to that population. But, extending this inference to the population of grammars as a whole is speculative at best. This deficiency, however, is minor; the causal relationship is strong even though it applies only to ANTLR grammars.

VII. Threats to Validity

We focused on threats to the conclusion, construct, internal, and external validity as detailed by Wohlin et al. [29]. We have identified no threats to the conclusion and construct validity of this study, but there is a threat to the internal validity of this study. Specifically, we selected grammars from the ANTLR repository which limits the grammars available as well as the formats in which they are implemented. This is a threat due to the under-representation of the domain. Additionally, since we restrict our internal representations to BNF and ANTLR 4 and do not include the ability to utilize TXL, SDF or other grammar formats, there is a threat to external validity.

VIII. Conclusions and Future Work

In this paper we have developed an algorithmic approach to merging programming language grammars together. The goal of this approach is to facilitate the automatic creation of Island Grammars for use in the development of multi-lingual software analysis tools. We evaluate this approach through two experiments on combined grammars selected from the ANTLR grammars-v4 repository. These experiments evaluated the effects of this approach on changes to Halstead Effort and McCabe Cyclomatic Complexity of the grammar pairs as they are merged. The results show that the merging approach reduces the Halstead Effort and Cyclomatic Complexity when taking both grammar size and similarity threshold into consideration.

The results of this study are promising and present a potentially fruitful avenue towards the development of an automated approach to Island Grammar generation. Island Grammars have been shown to be an extremely useful approach for the development of multi-lingual analysis tools, but can be time-consuming to develop.

Immediate avenues for future work are as follows. We intended to conduct further studies to improve the results herein by expanding the study to grammars selected from the GrammarZoo [30] collection. As part of this we intend to extend the capabilities of this approach to incorporate TXL and SDF grammars which will also reduce threats to validity identified in Section VII. The results herein indicate the promise of this approach and we intend to integrate it with ongoing work to develop tools which will support the automated construction of island grammars for use in static analysis and quality measurement of software systems.

Acknowledgements

This research is supported by funding from the Ronald E. McNair Post Baccalaureate Achievement Program at Idaho State University, which is sponsored by the Department of Education (P217A170169).

References

[1] Z. Mushtaq, G. Rasool, and B. Shehzad, "Multilingual Source Code Analysis: A Systematic Literature Review," *IEEE Access*, vol. 5, pp. 11 307–11 336, 2017.

[2] R. Kirkov and G. Agre, "Source Code Analysis – An Overview," p. 19.

[3] N. Synytskyy, J. R. Cordy, and T. R. Dean, "Robust multilingual parsing using island grammars," in *Proceedings of the 2003 Conference of the Centre for Advanced Studies on Collaborative Research*. IBM Press, 2003, pp. 266–278.

[4] V. R. B. G. Caldiera and H. D. Rombach, "The goal question metric approach," *Encyclopedia of software engineering*, pp. 528–532, 1994.

[5] M. Haoxiang, *Languages and Machines: An Introduction to the Theory of Computer Science*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co. Inc., 1988.

[6] L. Moonen, "Generating robust parsers using island grammars," in *Proceedings Eighth Working Conference on Reverse Engineering*, Oct. 2001, pp. 13–22.

[7] A. V. Deursen and T. Kuipers, "Building documentation generators," in *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*, Aug. 1999, pp. 40–49.

[8] L. Moonen, "Lightweight Impact Analysis using Island Grammars." in *IWPC*. Citeseer, 2002, pp. 219–228.

[9] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2008, pp. 308–318.

[10] A. Bacchelli, A. Cleve, M. Lanza, and A. Mocci, "Extracting structured data from natural language documents with island parsing," in *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference On*. IEEE, 2011, pp. 476–479.

[11] S. Klusener and R. Lammel, "Deriving tolerant grammars from a base-line grammar," in *International Conference on Software Maintenance*. IEEE, 2003, pp. 179–188.

[12] A. Goloveshkin and S. Mikhalkovich, “Tolerant parsing with a special kind of «Any» symbol: The algorithm and practical application,” *Proceedings of the Institute for System Programming of the RAS*, vol. 30, no. 4, pp. 7–28, 2018.

[13] J. Kurš, M. Lungu, R. Iyadurai, and O. Nierstrasz, “Bounded seas,” *Computer languages, systems & structures*, vol. 44, pp. 114–140, 2015.

[14] P. Linos, W. Lucas, S. Myers, and E. Maier, “A Metrics Tool for Multi-language Software,” p. 10.

[15] H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque, “PIT: A practical mutation testing tool for Java (demo),” in *Proceedings of the 25th International Symposium on Software Testing and Analysis - ISSTA 2016*. Saarbr&#252;cken, Germany: ACM Press, 2016, pp. 449–452.

[16] I. Clarke, “Experimenting with “mutation testing” and Kotlin,” https://medium.com/@s4n1ty/experimenting-with-mutation-testing-and-kotlin-b515d77e85b5, Oct. 2016.

[17] A. Janes, D. Piatov, A. Sillitti, and G. Succi, “How to Calculate Software Metrics for Multiple Languages Using Open Source Parsers,” in *Open Source Software: Quality Verification*, E. Petrinja, G. Succi, N. El Ioini, and A. Sillitti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, vol. 404, pp. 264–270.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge Massachusetts: The MIT Press, 2001.

[19] J. F. Power and B. A. Malloy, “A metrics suite for grammar-based software,” *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 16, no. 6, pp. 405–426, Nov. 2004.

[20] T. Parr, *The Definitive ANTLR 4 Reference*, ser. The Pragmatic Programmers. Dallas, Texas: The Pragmatic Bookshelf, 2012, oCLC: ocn802295434.

[21] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Berlin; London: Springer, 2011, oCLC: 750954916.

[22] D. C. Montgomery, *Design and Analysis of Experiments*, eighth edition ed. Hoboken, NJ: John Wiley & Sons, Inc, 2013.

[23] H. Levene, “Robust tests for equality of variances,” *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*, vol. 2, pp. 278–292, 1960.

[24] T. W. Anderson and D. A. Darling, “A test of goodness of fit,” *Journal of the American Statistical Association*, vol. 49, no. 268, p. 765, Dec. 1954.

[25] J. J. Higgins, *An Introduction to Modern Nonparametric Statistics*. Pacific Grove, CA: Brooks/Cole, 2004.

[26] C. W. Dunnett, “A Multiple Comparison Procedure for Comparing Several Treatments with a Control,” *Journal of the American Statistical Association*, vol. 50, no. 272, pp. 1096–1121, Dec. 1955.

[27] R. G. D. Steel, “A Multiple Comparison Sign Test: Treatments Versus Control,” *Journal of the American Statistical Association*, vol. 54, no. 288, p. 767, Dec. 1959.

[28] A. R. Jonckheere, “A Distribution-Free k-Sample Test Against Ordered Alternatives,” *Biometrika*, vol. 41, no. 1/2, p. 133, Jun. 1954.

[29] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[30] V. Zaytsev, “Grammar Zoo: A corpus of experimental grammarware,” *Science of Computer Programming*, vol. 98, pp. 28–51, Feb. 2015.