

An Object-Oriented Metrics Suite for Ada 95

William W. Pritchett IV
DCS Corporation
1330 Braddock Place
Alexandria, VA 22314
703.683.8430 x726
wpritch@dcscorp.com

1. ABSTRACT

Ada 95 added object-oriented programming capabilities to the existing Ada standard. The object-oriented paradigm results in new relationships within and among software modules. Traditional software product metrics, i.e., those metrics developed for functionally oriented software, do not account for these new relationships and may be insufficient for use in object-oriented development. This paper addresses this deficiency by 1) defining a set of object-oriented metrics tailored to Ada 95 software, and 2) providing empirical evidence of their validity. This set of metrics includes product measures quantifying the class-related software attributes of size, coupling, cohesion, and complexity. The results indicate that many of the metrics examined do correlate to the number of revisions made to the classes during development.

2. INTRODUCTION

The measurement and evaluation of internal software attributes is increasing being seen as a way to improve overall software product quality. Measurement also plays an important part in process improvement as described in the Software Engineering Institute's Capability Maturity Model (CMM). Fenton describes two general purposes for

software measurement, prediction, and assessment [1]. For example, quality assurance personnel use measures to predict the number of defects in a module. Managers, on the other hand, may rely on such measures to assess programmer productivity.

A common practice in software development is to measure external and internal product attributes to identify potential fault-prone modules, which may help in prioritizing testing resources. For functionally developed software, traditional product measures include cyclomatic complexity [2], Halstead measures [3], and lines of code.

These traditional measures, however, may not be appropriate for object-oriented software. Tegarden et al. have found that "assumptions relating to program size and programmer productivity in structured systems do not apply directly to OO systems. Second, the traditional metrics do not address the structural aspects of OO systems" [4]. Further, Brooks has found that "metrics like McCabe complexity to not appear to be useful for OO development and that it seems necessary to find useful measures for polymorphism, inheritance, and the cohesion between class attributes" [5].

Given this lack of suitability of these traditional measures for use in object-oriented software, several researchers have proposed new measures that are specific to object-oriented software. Chidamber and Kemerer have proposed the most popular of these measures [6]. The proposed metrics include Weighted Methods Per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Response for a Class (RFC), and Lack of Cohesion in Methods (LCOM). Though criticized for a number of reasons [7][8], these metrics have become accepted and their collection is supported by a number of commercially available tools.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page.
To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGAda 2001 09/01 Bloomington, MN, USA
© 2001 ACM 1-58113-392-8/01/0009...\$5.00

A important aspect of any measure is its validity with respect to either how well the measure assesses an attribute of an entity, or how well the measure is able to predict some attribute of a future entity. Several studies have been conducted in order to validate the Chidamber and Kemerer (C-K) measures [9][10][11][12]. These studies, however, neither defined these metrics in the context of object-oriented Ada 95 software nor empirically validated the measures using object-oriented Ada 95 software. This paper addresses this deficiency by 1) defining a set of object-oriented metrics tailored to Ada 95 software, and 2) providing empirical evidence of their validity.

2.1 METRICS FOR Ada 95

The revision of the Ada programming language, ISO/IEC 8652:1995, included new language features supporting object-oriented programming. Though previously supporting the principles of information hiding and encapsulation, Ada now includes support for inheritance, polymorphism, and the notion of a class. A class in Ada consists of a tagged record type and the primitive operations defined for that type. This can be combined with the package construct and the private region to support encapsulation and information hiding.

With the new features of the language, the object-oriented measures defined by C-K can now be applied to Ada 95 software. The author, in a previous work, defined how the measures could be calculated in Ada 95 [13]. The remainder of this section will include the definitions for each measure, how it is applied to Ada 95, a brief rationale for the measure, and a hypothesis relating to the measure's ability to predict future faults.

2.2 Depth of Inheritance

Inheritance is programming by extension, which is being able to add additional attributes and operations to an existing class to extend the functionality for a new purpose or abstraction. The Depth of Inheritance measure (C-K DIT) is defined to be the level of the class in the tagged type inheritance tree, with the root class being zero. This measure is important because changes to the parent class may impact the descendents. Additionally, the bulk of the attributes and operations should be defined in the root, as it is the most general class in the tree. Therefore, the majority of the faults should appear higher in the class hierarchy. This rationale leads to the following hypothesis:

Hypothesis: The number of faults decreases as the depth increases.

2.3 Total Number of Children

The Total Number of Children measure (C-K NOC) is a count of all descendents for a class. This measure is important because classes with large numbers of descendents are harder to modify because changes may affect the descendents. Classes with many descendents

tend to be larger as they encapsulate all of the common attributes and operations for a large number of classes. This rationale leads to the following hypothesis:

Hypothesis: The number of faults increases as the number of children increases.

2.4 Total Attributes

The Total Attributes measure is a count of the total number of attributes; both inherited and locally defined, for a class. The attributes for an Ada class are defined to be the components in a tagged record. This measure is important because a large number of attributes may indicate a poor design (improper abstraction), or that the class is complicated and therefore more fault-prone. This leads to the following hypothesis:

Hypothesis: The number of faults increases as the total number of attributes increases.

2.5 Local Attributes

The Local Attributes measure is a count of the locally defined attributes in a class (those components immediately defined in the tagged record). This measure, though similar in nature to the Total Attributes measure, is important because it characterizes the difference (delta) between the number of total attributes defined and local attributes defined. The size of the delta may indicate a class that is "complex" (high ratio of local attributes to total attributes), or tightly coupled to its ancestors (low ratio of local attributes to total attributes). This leads to the following hypothesis:

Hypothesis: The number of faults increases as the total number of locally defined attributes increases.

2.6 Total Operations

Total Operations is a count of the total number of primitive operations defined for the class. This includes those operations inherited, but not overridden and those operations locally defined. This measure is important because functionality increases as the number of operations increases, thus increasing the likelihood of faults. This leads to the following hypothesis:

Hypothesis: The number of faults increases as the total number of operations increases.

2.7 Local Operations

The Local Operations measure is a count of the total number of primitive operations locally defined for the class (defined in the same package in which the tagged type is defined). Though similar in nature to the Total Operations measure, the number of local operations is important because it characterizes the difference (delta) between the number of total operations defined and local operations defined. The size of the delta may indicate a class that is "complex" (high ratio of local operations to total

operations), or tightly coupled to its ancestors (low ratio of local operations to total operations). This leads to the following hypothesis:

Hypothesis: The number of faults increases as the total number of locally defined operations increases.

2.8 Overridden Operations

The Overridden Operations measure is a count of the number of operations defined in an ancestor, but overridden by the class. Overriding operations may complicate the understandability of a class and may also provide an opportunity for errors. This leads to the following hypothesis:

Hypothesis: The number of faults increases as the number of overridden operations increases.

2.9 Class-Wide Operations

The Class-Wide Operations measure is a count of the number of class-wide operations defined for the class. Class-wide operations are unique to Ada and, therefore, little is known about its effects on reliability and maintainability. Class-wide operations must account for all possible descendents and may therefore be more prone to faults. This leads to the following hypothesis:

Hypothesis: The number of faults increases as the number of class-wide operations increases.

2.10 Message Passing Coupling

The Message Passing Coupling measure (CK=MPC) is a count of the total number of calls (functions and procedures) made to external units. Different calls to the same routine are counted separately. This measure quantifies the extent to which the class is coupled to other classes. A high level of coupling is generally considered problematic in that changes in a class may affect all of the classes to which it is coupled. This leads to the following hypothesis:

Hypothesis: The number of faults increases as the message passing coupling increases.

2.11 Attributes of Abstract Data Types

The Attributes of Abstract Data Types (ADTs) measure is a count of the total number of record components whose type is an ADT. Two attributes having the same type are counted twice. It is essentially a count of the attributes whose type is not an Ada predefined type or a derivative of a predefined type. The number of ADTs referenced is measure of coupling between the class and other classes. This leads to the following hypothesis:

Hypothesis: The number of faults increases as the number of Attributes of ADTs increases.

2.12 Abstract Data Types Referenced

The Abstract Data Types Referenced measure is a count of the unique number of ADTs referenced. This count is similar to the Attributes of ADTs, however, attributes having the same type are only counted once. Like the Attributes of ADTs measure, it is a measure of the coupling between classes. This leads to the following hypothesis:

Hypothesis: The number of faults increases as the number of ADTs referenced increases.

2.13 Class Cohesion

The Class Cohesion measure (C-K LCOM) is a measure of the lack of cohesion of methods. This is computed by subtracting from the number of pairs of operations without shared instance variables the number of pairs of operations with shared variables. The measure is zero if the subtraction results in a negative number. This measure is important as a high lack of cohesion indicates the classes operations do not operate on the attributes and, therefore, is a poor abstraction/poorly designed class. Thus, the class is more likely to contain faults as the operations and attributes have little to do with each other and the implementation of the class may be complex. This leads to the following hypothesis:

Hypothesis: The number of faults increases as the class cohesion measure (lack of cohesion) increases.

2.14 Max Cyclomatic Complexity

The Max Cyclomatic Complexity measure is the maximum cyclomatic complexity of the individual complexities of the locally defined operations for the class. The complexities of the individual operations are important and should be kept to a minimum as in functionally developed software. The maximum complexity is an indication of the worst case complexity for the class, and when developing and maintaining the class as a whole, should be considered when evaluating the quality of the class. This leads to the following hypothesis:

Hypothesis: The number of faults increases as the maximum cyclomatic complexity increases.

3. EMPIRICAL STUDY

The empirical study follows the framework suggested by Basili, et. al. and includes the phases of definition, planning, operation, and interpretation [14].

3.1 Study Definition

The motivation of this study was to improve the unit testing process in order to identify those Ada 95 classes that are more prone to faults. The study focused on identifying those metrics that are predictors of faults in Ada 95 software developed using an object-oriented approach from the perspective of the developer. A reusable library for embedded graphics consisting of 94 classes and

approximately 36,000 lines of code (semicolons) was examined. Two experience programmers with significant Ada 95 and embedded graphics experience developed the library.

3.2 Study Planning and Operation

The study consisted of using the AdaSTAT static analysis tool to automatically collect metrics data for each of the classes. The metrics collected serve as the *independent variable* and include Maximum Cyclomatic Complexity (Max V(G)), Depth of Inheritance (DIH), Total Children (TC), Abstract Data Types Referenced (ADT-R), Attributes of Abstract Data Types (A-ADT), Message Passing Coupling (MPC), Class Cohesion (CC), Overridden Operations (OO), Total Operations (TO), Local Operations (LO), Class-wide Operations (CO), Total Attributes (TA), and Local Attributes (LA).

The *dependent variable* used was a count of the number of revisions during development. Graves et. al. found this to be an effective model for measuring fault potential as they determined that a module’s expected number of faults is proportional to the number of times it has been changed [15]. The revision count for this study was collected using the output of the Unix Source Code Control System (SCCS) utility.

Once collected, the data was entered into the Minitab statistical software package and descriptive statistics for each of the metrics were collected. Next, the Pearson Correlation Coefficients for each of the metrics considered against the number of revisions for each file was computed. This coefficient is a measure of the degree of linear relationship between the two variables. If one variable increases as the other increases, the coefficient is positive. If one variable decreases as the other increases, then the coefficient is negative.

Lastly, a univariate straight-line regression analysis was performed on each of the metrics versus the revision count. Regression analysis addresses the “formulation of mathematical models that depict relationships for the purpose of prediction and other statistical inferences” [16]. The regression analysis results in a model of the form

$$Y = \beta_0 + \beta_1 X + \epsilon$$

where Y is the response variable (number of revisions), X is the predictor (the metric), β_0 and β_1 are the regression coefficients, and ϵ is an error term. A non-zero β_1 indicates the response variable changes as the predictor changes. Using this approach, one can test the null hypothesis $H_0: \beta_1=0$ indicating no relationship against an alternative ($\beta_1 \neq 0$), which indicates a relationship. To further illustrate the regression relationship, the results include scatter plots of the metrics versus number of revisions overlaid with the fitted straight-line.

3.3 Study Limitations

The study has notable limitations that may restrict any broad-reaching conclusions. The correlation of SCCS revision data to faults may not be entirely valid. Under the process by which the modules were developed, revisions to the software could occur for a variety of reasons other than defects. Examples include changes in requirements or files being checked into SCCS without any modifications made at all. Additionally, revisions are counted at the file level and files may contain multiple packages and packages may contain multiple classes making it difficult to correlate a revision to a particular class.

Furthermore, this study only examined the software from a single project with a limited number of developers. No attempt was made to control for other variables that may have affected the results such as the skill/experience level of the developers or the process by which the individuals developed their modules.

3.4 Results

Table 1 contains the descriptive statistics for each of the metrics considered.

Table 1. Descriptive Statistics

Measure	Min	Max	Median	Mean	Std. Dev
Max V(G)	1	38	3	5.5	6.52
DIH	0	1	1	0.92	0.26
TC	0	34	0	0.85	4.43
ADT-R	0	11	1	1.81	2.62
A-ADT	0	17	1	2.12	3.28
MPC	0	17	1	0.93	1.84
CC	0	0.88	0	0.33	0.36
OO	0	55	6	7.13	9.36
TO	2	83	9	19.31	17.77
LO	2	55	7	8.77	10.85
CO	0	47	0	5.22	9.45
TA	0	24	7	10.93	6.23
LA	0	16	1	2.17	3.48

Table 2 contains the Pearson Correlation Coefficient for each of the metrics considered versus the number of revisions. The table also includes the p-value (significance) for each of the coefficients.

Table 2. Pearson Correlation Coefficients

Measure	Correlation Coefficient	P
Max V(G)	0.457	0.000
DIH	-0.547	0.000
TC	0.459	0.000
ADT-R	0.504	0.000

Measure	Correlation Coefficient	P
A-ADT	0.457	0.000
MPC	-0.041	0.695
CC	0.311	0.002
OO	-0.22	0.833
TO	0.309	0.002
LO	0.424	0.000
CO	0.068	0.513
TA	0.225	0.029
LA	0.610	0.000

Table 3 contains the Regression Coefficient (β_1) for each of the metrics considered.

Table 3. Regression Coefficients

Measure	β_1	p
Max V(G)	7.72	0.004
DIH	-35.15	0.000
TC	1.76	0.000
ADT-R	3.27	0.000
A-ADT	2.36	0.000
MPC	0.377	0.695
CC	14.5	0.002
OO	-0.04	0.833
TO	0.295	0.002
LO	0.662	0.000
CO	0.122	0.513
TA	0.613	0.029
LA	2.98	0.000

4. INTERPRETATION OF RESULTS

4.1 Max Cyclomatic Complexity

Both the regression analysis and the correlation coefficient provide evidence that as the number of revisions increases as the maximum cyclomatic complexity increases. It is therefore possible to reject the regression analysis null hypothesis, $H_0: \beta_1=0$, at $\alpha=0.01$ and conclude there is a relationship between the maximum cyclomatic complexity for a class and defects.

This result is consistent with previous work that correlates cyclomatic complexity with defects for functionally developed software [17][18][19]. The accepted criteria of keeping the cyclomatic complexity low for a subprogram still holds for object-oriented software.

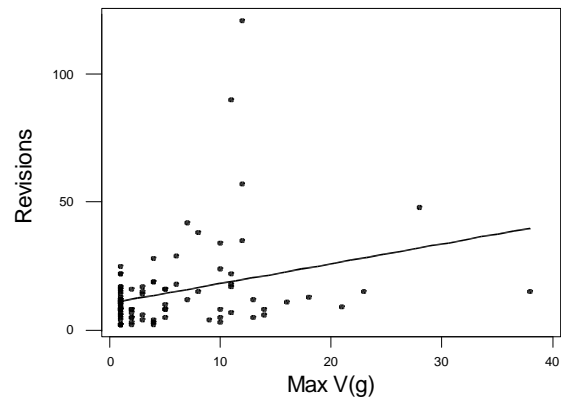


Figure 1. Scatter Plot of Max V(G) vs. Revisions

4.2 Depth of Inheritance

Both the regression analysis and the correlation coefficient provide evidence that the number of revisions decreases as the depth of inheritance increases. It is therefore possible to reject the regression analysis null hypothesis (there is no relationship), $H_0: \beta_1=0$, at $\alpha=0.01$ and conclude there is a relationship between the depth of inheritance for a class and defects.

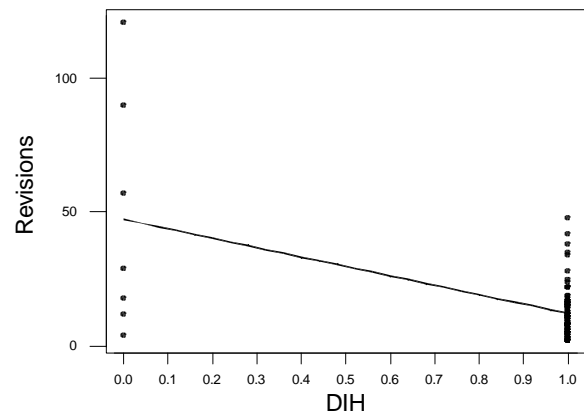


Figure 2. Scatter Plot of DIH vs. Revisions

The relationship confirms the original hypothesis that the number of revisions should decrease as the depth of inheritance increases. This is probably due to the fact that the root classes are large and encapsulate greater functionality and have a greater potential for faults.

4.3 Number of Children

Both the regression analysis and the correlation coefficient provide evidence that as the number of revisions increases as the total number of children increases. It is therefore possible to reject the regression analysis null hypothesis, $H_0: \beta_1=0$, at $\alpha=0.01$ and conclude there is a relationship between the total number of children for a class and defects.

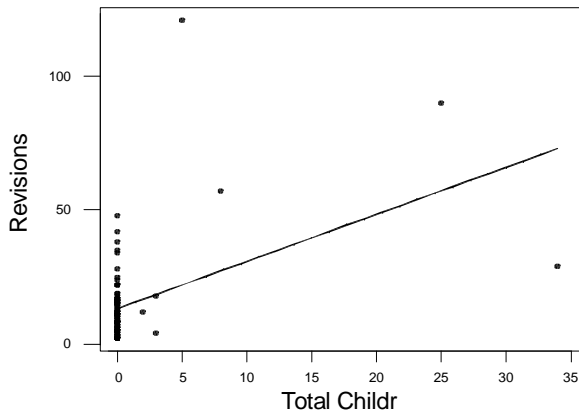


Figure 3. Scatter Plot of Total Children vs. Revisions

This relationship confirms the original hypothesis and is possibly due to the fact that the parent class has to account for (be an abstraction of) all of its children. As children are added, the parent may have to change to reflect the new children if the design was not originally correct.

4.4 Abstract Data Types Referenced

Both the regression analysis and the correlation coefficient provide evidence that as the number of revisions increases as the number of abstract data types referenced increases. It is therefore possible to reject the regression analysis null hypothesis, $H_0: \beta_1=0$, at $\alpha=0.01$ and conclude there is a relationship between the number of abstract data types referenced for a class and defects.

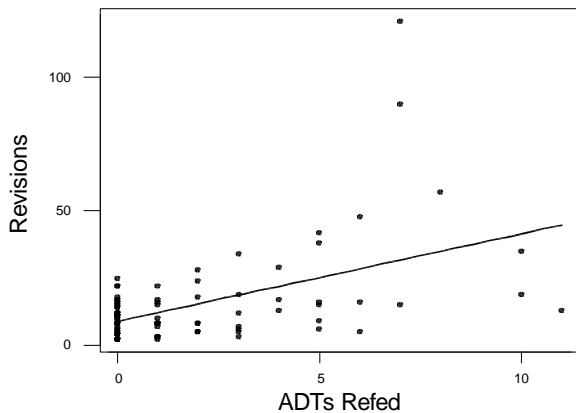


Figure 4. Scatter Plot of ADTs Referenced vs. Revisions

This relationship confirms the original hypothesis and could be due to the fact that each ADT referenced represents a couple with another class or type. It is generally accepted that coupling should be minimized.

4.5 Attributes of Abstract Data Types

Both the regression analysis and the correlation coefficient provide evidence that as the number of revisions increases as the number of attributes whose type is an ADT

increases. It is therefore possible to reject the regression analysis null hypothesis, $H_0: \beta_1=0$, at $\alpha=0.01$ and conclude there is a relationship between the number of abstract data types referenced for a class and defects.

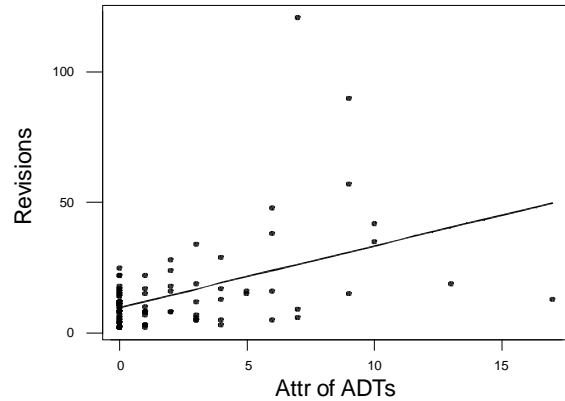


Figure 5. Scatter Attributes of ADTs vs. Revisions

This relationship confirms the original hypothesis and may be explained by the fact that each attribute that references an ADT represents a couple with another class or type. It is generally accepted that coupling should be minimized.

4.6 Message Passing Coupling

Neither the regression analysis nor the correlation coefficient provides evidence that as the number of revisions increases as message passing coupling increases. The regression analysis null hypothesis, $H_0: \beta_1=0$, cannot be rejected at $\alpha=0.01$.

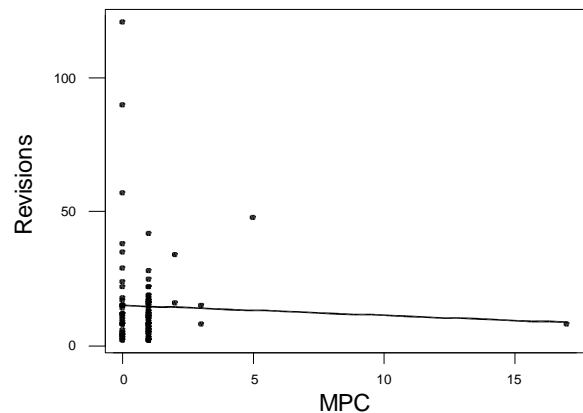


Figure 6. Scatter Plot of MPC vs. Revisions

This does not confirm the original hypothesis that defects increase as the MPC increases. Though the data does not support this, each operation invoked on another object represents a couple with that object and coupling in general should be kept to a minimum. This metric warrants further investigation as to its effectiveness as a predictor of fault-prone classes.

4.7 Class Cohesion

Both the regression analysis and the correlation coefficient provide evidence that as the number of revisions increases as the class cohesion measure increases. It is therefore possible to reject the regression analysis null hypothesis, $H_0: \beta_1=0$, at $\alpha=0.01$ and conclude there is a relationship between the cohesion of a class and defects.

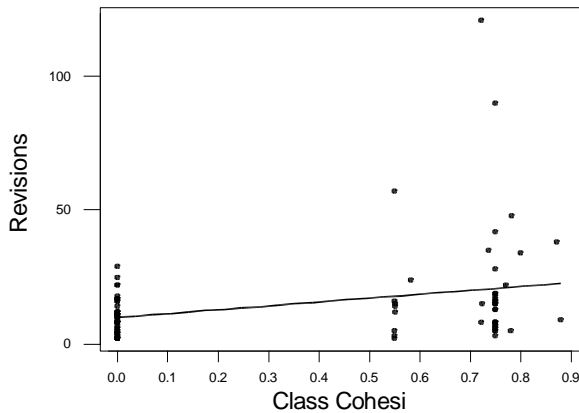


Figure 7. Scatter Plot of Class Cohesion vs. Revisions

This supports the original hypothesis that cohesive classes are less prone to faults as this is a measure of the *lack* of cohesion of methods.

4.8 Overridden Operations

Neither the regression analysis nor the correlation coefficient provides evidence that as the number of revisions increases as the number of overridden operations increases. It is therefore not possible to reject the regression analysis null hypothesis, $H_0: \beta_1=0$, at $\alpha=0.01$.

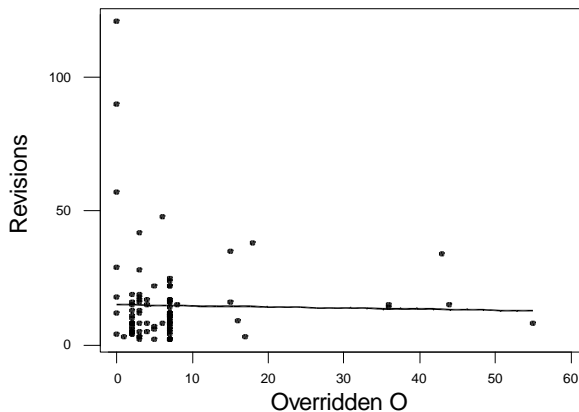


Figure 8. Scatter Plot of Overridden Ops vs. Revisions

This does not support the original hypothesis that faults increase as the number of overridden operations increases. This could be due to the limitations of the study. Though potentially not useful for predicting fault-prone classes, this metric could have an impact on understandability as care

must be taken in deciding which operation is actually called in a dispatching operation.

4.9 Total Operations

Both the regression analysis and the correlation coefficient provide evidence that as the number of revisions decreases as the number of total operations increases. It is therefore possible to reject the regression analysis null hypothesis, $H_0: \beta_1=0$, at $\alpha=0.01$ and conclude there is a relationship between the total number of operations for a class and defects.

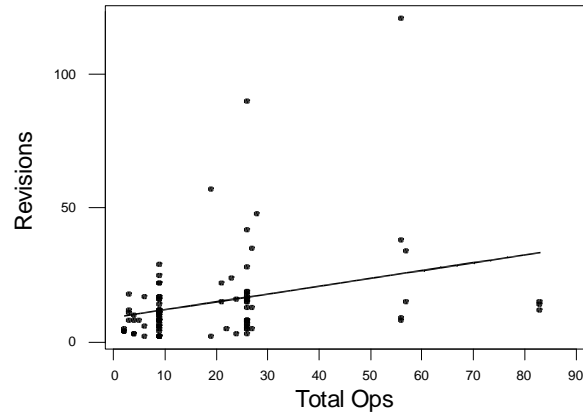


Figure 9. Scatter Plot of Total Operations vs. Revisions

This supports the original hypothesis that the number of faults increases as the total number of operations increases. This could be due to the fact that the size of a class increases as the number of operations increases (more physical lines of code), thus increasing the likelihood of errors.

4.10 Local Operations

Both the regression analysis and the correlation coefficient provide evidence that as the number of revisions decreases as the number of locally defined operations increases. It is therefore possible to reject the regression analysis null hypothesis, $H_0: \beta_1=0$, at $\alpha=0.01$ and conclude there is a relationship between the total number of locally defined operations for a class and defects.

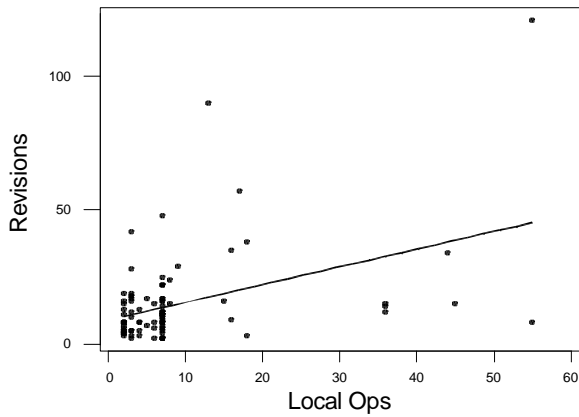


Figure 10. Scatter Plot of Local Ops vs. Revisions

This supports the original hypothesis that the number of faults increases as the total number of local operations increases. This could be due to the fact that the size of a class increases as the number of operations increases, thus increasing the likelihood of errors. It should also be noted that the number of local operations appears to have a stronger correlation with revisions than total number of operations. Given that they have a strong correlation with each other, it may be sufficient to measure only local operations.

4.11 Class-Wide Operations

Neither the regression analysis nor the correlation coefficient provides evidence that as the number of revisions increases as the number of class-wide operations increases. It is therefore not possible to reject the regression analysis null hypothesis, $H_0: \beta_1=0$, at $\alpha=0.01$.

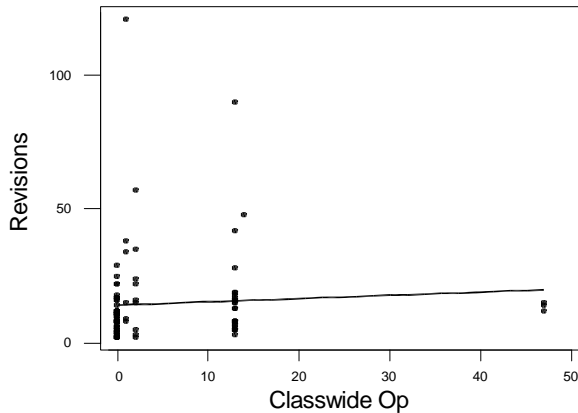


Figure 11. Scatter Plot of Class-wide Ops vs. Revisions

This does not support the original hypothesis that faults increase as the number of class-wide operations increase. This could be attributed to the limitations of the study, specifically that few classes in the library use class-wide operations. This metric may also be better suited for understanding and predicting maintainability.

4.12 Total Attributes

Neither the regression analysis nor the correlation coefficient provides sufficient evidence that the number of revisions increases as the total number of attributes increases. It is therefore not possible to reject the regression analysis null hypothesis, $H_0: \beta_1=0$, at $\alpha=0.01$. (Though it is possible to reject the null hypothesis at $\alpha=0.05$)

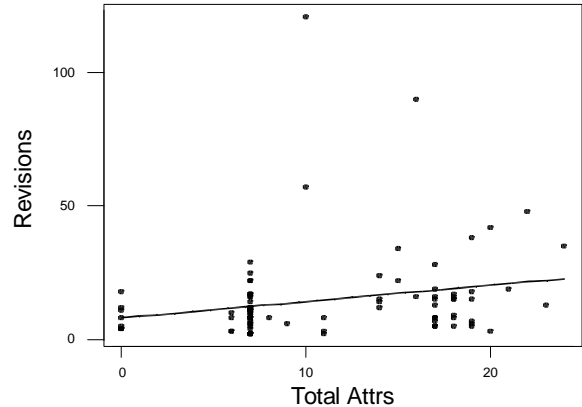


Figure 12. Scatter Plot of Total Attributes vs. Revisions

This does not support the original hypothesis that faults increase as the total number of attributes increases. This could be attributed to the limitations of the study. Also, notice from the scatter plot the existence of three outliers that had a strong influence on the outcome.

4.13 Local Attributes

Both the regression analysis and the correlation coefficient provide evidence that as the number of revisions increases as the number of locally defined attributes increases. It is therefore possible to reject the regression analysis null hypothesis, $H_0: \beta_1=0$, at $\alpha=0.01$ and conclude there is a relationship between the number of locally defined attributes for a class and defects.

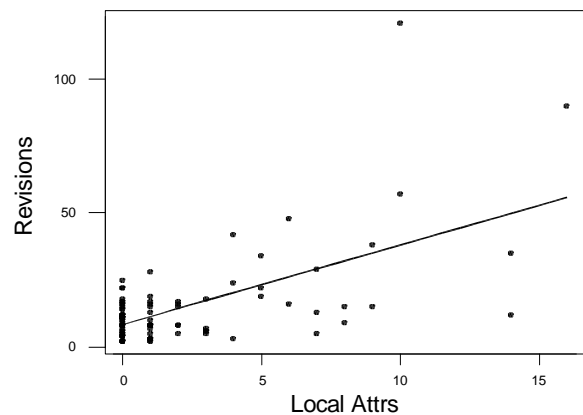


Figure 13. Scatter Plot of Local Attributes vs. Revisions

This supports the original hypothesis that faults increase with the number of locally defined attributes. This metric is also highly correlated with attributes of abstract data types, with the main difference being that locally defined attributes includes predefined types (and derivations of predefined types). Given that attributes of ADTs are more correlated to revisions, it may be sufficient to measure only attributes of ADTs.

5. SUMMARY

This paper has proposed a set of metrics for Ada 95 software developed using object-oriented techniques. This paper includes the results of empirically validating the proposed metrics as being predictors of fault-prone classes. Based on the experimental goals, this paper set forth a hypothesis about each of the metrics with regard to being predictors of fault prone classes.

There is sufficient evidence to conclude that the following OO metrics are predictors of faults in classes: Maximum Cyclomatic Complexity, Depth of Inheritance, Number of Children, ADTs Referenced, Attributes of ADTs, Total Operations, Local Operations, and Local Attributes. There was not sufficient evidence to conclude that the following metrics are predictors of fault-prone classes: message passing coupling, class cohesion, overridden operations, and class-wide operations. These metrics require addition investigation to determine their usefulness.

Finally, given that Ada 95 is a hybrid language supporting both OO and non-OO development, a mix of traditional and OO metrics may be appropriate.

6. FUTURE WORK

To better assess the metrics under consideration, the study should be replicated to include more classes from more projects. The study should also be performed using actual defect data instead of SCCS revision data to get a more accurate model of fault prediction. More work should also be done to assess which of the metrics are better at predicting faults than others. This may help better focus measurement resources.

Furthermore, the set of metrics considered is not, by any means, all-inclusive. Additional metrics that better capture the attributes and relationships of object-oriented Ada software should be derived and validated.

Lastly, a multivariate model that predicts software maintainability based on object-oriented metrics should be developed. This model would augment the Maintainability Index developed by Oman et al. [20] for functionally developed software.

7. REFERENCES

[1] N. Fenton, *Software Metrics: A Rigorous Approach*, Chapman & Hall, London, 1991.

- [2] T. McCabe, "A Complexity Measure," *IEEE Transactions of Software Engineering*, vol. 2, Dec. 1976, pp. 308-320.
- [3] M. Halstead, *Elements of Software Science*, New York: Elsevier, North-Holland, 1977.
- [4] D. Tegarden, S. Sheetz, and D. Monarchi. "Effectiveness of Traditional Software Metrics for Object-Oriented Systems," *Twenty-Fifth Hawaiian International Conference on System Sciences - HICSS-25*, Jan. 92.
- [5] I. Brooks, "Object-Oriented Metrics Collection and Evaluation with a Software Process," *Proc. OOPSLA '93 Workshop Processes and Metrics for Object-Oriented Software Development*, Washington, D.C., 1993.
- [6] S. Chidamber and C. Kemerer, "A Metric Suite for Object-Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, pp476-493, 1994.
- [7] N. Churcher and M. Shepperd, "Comments on 'A Metrics Suite for Object Oriented Design,'" *IEEE Transactions on Software Engineering*, vol. 21, no. 3, March 1995, pp. 263-265.
- [8] M. Hitz and B. Montazeri, "Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective," *IEEE Transactions on Software Engineering*, vol. 22, no. 4 April 1996, pp. 267-271.
- [9] M. Cartwright and M. Shepperd, "An Empirical Investigation of an Object-Oriented Software System," *IEEE Transactions on Software Engineering*, Vol. 26, No. 8, Aug. 2000, pp. 786-796.
- [10] V. Basili, L. Brian, and W. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators", *IEEE Transactions on Software Engineering*, Vol. 22, No. 10, October 1996, pp. 751-761.
- [11] L. Briand, C. Bunse, and J. Daly, "A Controlled Experiment for Evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs," *IEEE Transactions on Software Engineering*, Vol. 27, No. 6, June 2001, pp. 513-530.
- [12] L. Brian, S. Morasca, and V. Basili, "Defining and Validating Measures for Object-Based High-Level Design," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, Sept/Oct 1999, pp. 722-743.
- [13] W. Pritchett, "Applying Object-Oriented Metrics to Ada 95," *Ada Letters*, Vol. XVI, No. 5, Sep/Oct 1996, pp. 48-58.
- [14] V. Basili, R. Selby, and D. Hutchens, "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 12, no.7, July 1986, pp. 733-743.

- [15] Graves, Karr, Marron, and Siy, "Predicting Fault Incidence Using Software Change History," *IEEE Transactions of Software Engineering*, Vol. 26, No. 7, July 2000, pp. 653-661.
- [16] G. Bhattacharyya and R. Johnson, *Statistical Concepts and Methods*, Wiley and Sons, New York, New York. 1977.
- [17] Walsh, T. "A Software Reliability Study Using a Complexity Measure," *AFIPS Conference Proceedings*, AFIPS Press, 1979.
- [18] Henry, S., Kafura, D., and Harris, K. "On the Relationship Among Three Software Metrics," *1981 ACM Workshop/Symposium on Measurement and Evaluation of Software Quality*, March 1981.
- [19] Ward, W., "Software Defect Prevention Using McCabe's Complexity Metric," *Hewlett-Packard Journal*, April 1989.
- [20] K. Welker and P. Oman, "Software Maintainability Metrics Models in Practice," *Crosstalk*, Nov/Dec. 1995.