2011

Object-oriented runtime software quality analysis

Ruchira Mathur

OBJECT-ORIENTED
RUNTIME SOFTWARE QUALITY ANALYSIS

by

RUCHIRA MATHUR

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
The Department of Computer Science
to
The School of Graduate Studies
of
The University of Alabama in Huntsville

HUNTSVILLE, ALABAMA
2011

In presenting this dissertation in partial fulfillment of the requirements for a doctoral degree from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department or the Dean of the School of Graduate Studies. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this dissertation.

Ruchira Mathur 5-24-2011

(Student signature) (Date)

DISSERTATION APPROVAL FORM

Submitted by Ruchira Mathur in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science and accepted on behalf of the Faculty of the School of Graduate Studies by the dissertation committee.

We, the undersigned members of the Graduate Faculty of The University of Alabama in Huntsville, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science.

5/12/11 (Date) Committee Chair

5-13-11

5-13-11

5-13-11

5/23/2011

Heggere S. Ranganath 5-12-11 Department Chair

College Dean

Rhonda Kay Gaede 8/7/11 Graduate Dean

ABSTRACT

The School of Graduate Studies
The University of Alabama in Huntsville

Degree Doctor of Philosophy College/Dept. Science/Computer Science

Name of Candidate Ruchira Mathur

Title Object-Oriented Runtime Software Quality Analysis

Traditional software metrics have been static metrics which measure various aspects of the software at compile time. In our research we have examined a very new field of software engineering research, that is, dynamic or runtime metrics that measure object oriented software quality in terms of cohesion and complexity when the code is executing. We define a suite of 2 dynamic complexity metrics and 4 dynamic cohesion metrics that were collected at runtime, compare them to Bug Data, Human Data, static metrics and (for cohesion) to another dynamic cohesion metric to validate them. Our results indicate that several of our dynamic metrics are actually measuring complexity or cohesion, as appropriate, and also shows there is some similarity between various dynamic metrics. Since the Dynamic metric research area is still in its infancy, we note the absence of established validation criteria in comparing Dynamic metrics to other static and dynamic metrics.

Abstract Approval: Committee Chair 5/12/11

Abstract Approval:

Department Chair Heggere S. Ranganath 5-12-11

Graduate Dean Chonda Kay Steede 8/7/11

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Letha Etzkorn, for all her help, guidance and support in this academic journey of a lifetime. It was a long process, but she helped me through it with her kind words and positive approach. Thanks are due to all my committee members for their help and guidance.

I would like to thank my husband, Pawan Mathur, and my children, Aditya Mathur and Arnav Mathur, for their loving support as I spend long hours doing my research and writing my thesis.

Thanks are due to my supervisors and all my co-workers for letting me do research and write my dissertation during work. I would like to especially thank Amy Ballard, Roger Berry and Dr. R. Buckelew for their encouragement and help.

I would also like to thank my parents, Mr. Ranjeet Asthana and my late mother, Mrs. Anjana Asthana, for raising me with the belief that nothing is impossible and instilling in me the desire to aim high.

Last but not the least, I would like to thank my mother-in-law, Mrs. Sudha Mathur, without whose help, I could not have accomplished my goal. She was always there to take care of me and our family when I was busy with school. Her help during this time was invaluable and she continues to remain a source of inspiration to me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

When considering overall software quality, there are two aspects that must be examined: run time performance (resource usage, throughput, etc.) and the "nonfunctional requirements" (reliability, reusability, flexibility, etc.). Runtime performance is measured at runtime on executable code, whereas nonfunctional requirements have traditionally been measured at compile time. In object-oriented software, for example, such non-functional requirements as reusability, flexibility, etc., are usually measured indirectly by looking at contributing lower level factors such as cohesion, coupling and complexity. These lower level factors are normally measured at the class level, on static source code at compile time.

It is clear that instantiated objects (compiled instances of individual classes at runtime) have runtime performance characteristics that are related to their status as objects. For example, each object requires resources, each object takes up CPU cycles. Further, however, the "nonfunctional requirements"-type characteristics of objects also have runtime equivalents. For example, the static coupling between one class and another would include static variable accesses that *could* occur in the source code. Runtime "coupling" between one object and another would also include *actual* variable accesses that *really* occurred at runtime. Similarly, message invocation across objects impacts runtime coupling [4].

However, this runtime boundary between static software quality measurement and runtime performance has not been heavily addressed before. The only prior work we have seen in this area is that of Mitchell and Power and Arisholm et al., who examined runtime cohesion and coupling [1] [22]. In our research, in addition to expanding the previous work in runtime cohesion, we also examine runtime complexity, which to our knowledge has not been addressed heretofore. This research will thus provide a better view of software quality than has been possible before.

CHAPTER 2

Background

Substantial work has been done in the area of software metrics in the past, but it has primarily been geared toward measuring static criteria: things which are measured at compile time [2] [3]. There has also been a related focus on work done trying to predict the quality of software [9]. Degree of coupling and degree of cohesion are just few of the measures that are currently being used to predict the quality of software. For example, a high degree of cohesion and a low degree of coupling indicates good quality software [3].

However the run time boundary between static object oriented metrics and run time performance metrics has not been heavily addressed before. We define "Runtime Boundary" as the consideration of the quality of instantiated objects of a class at runtime. This is not a performance metric because it is not measuring performance—it is measuring quality. This is not a traditional object-oriented metric because it is measuring runtime behavior rather than static qualities of code. It is in between performance metrics and static metrics indicating quality, which is why we call it a "boundary." The only prior work we have seen in this area is that of Mitchell and Power [23], and Arisholm et al. [1] who have defined runtime cohesion and runtime coupling metrics. We have not seen any previous research in the field of runtime complexity metrics. In our research we define 2 metrics to address runtime complexity.

2.1 Previous Work in Static Metrics

Many of the static object oriented metrics in use today are derived from the Chidamber and Kemerer suite of metrics [3]. They suggested the following static metrics:

- Weighted Metrics per class (WMC)—a sum of the complexities of the methods of a class. Often calculated with the complexity of each class set to unity; in that case, WMC is the number of methods in the class.
- Depth of Inheritance tree (DIT)—the depth of the class inheritance tree from base class to furthermost leaf.
- Number of children (NOC)—number of derived classes for which the current class is the parent.
- Coupling between object classes (CBO)—a count of the non-inheritance related couples between classes.
- Response for a class (RFC)—the number of methods called in a class, plus the number of methods called by those methods.
- Lack of cohesion in methods (LCOM)—a count of the members of the set of methods which access the same attribute. That is, if two methods access the same attribute, they represent one member in the set.

Henderson-Sellers [13] took the LCOM metric as defined by Chidamber and Kemerer and revised it to include the number of methods and instance variables. The Henderson-Sellers formulation has been shown to have advantages over the original Chidamber and Kemerer formulations [3] [13]. The metric Lack of Cohesion in Methods – Henderson-Sellers (LCOM_HS) has been defined as follows:

Consider a set of methods $\{M_i\}$ (i=1,…,m) accessing a set of attributes $\{A_j\}$ (j = 1,…,a). Let the number of attributes accessed by each method, M_i, be written as $\alpha(M_i)$ and the number of methods which access each datum be $\mu(A_j)$. Then

$$LCOM = [(1/a \Sigma_{j=1}^{a} \mu(A_j)) - m] / (1-m)$$ [13].

Other object-oriented metrics research includes Etzkorn et al. [11] who compared various existing cohesion metrics from a human oriented point of view. They gathered human data as well as did a statistical comparison to find the similarities in measurements.

2.2 Previous Work in Dynamic Metrics

Researchers have now realized the importance of taking measurements at run time and have pointed out that coupling and cohesion should be considered in the dynamic aspect also because message invocation across objects impacts system performance in run time [11][12][21][22][23][24].

Briand et al. [2] reason that it is difficult to evaluate dynamic metrics because it is difficult to determine the degree of coupling and cohesion between stand alone objects.

Mitchell and Power in their paper [23] have formulated definitions for coupling and cohesion at run time. They have defined the following run time metrics

Two coupling metrics:

- Dynamic Coupling between objects for a class A as the number of classes to which A is coupled at run time

- Degree of Dynamic Coupling within a given set of classes as the ratio of the sum of the number of accesses to methods or instance variables outside each class to the total number of accesses from these classes

Two cohesion metrics:

- Simple Dynamic LCOM as analogous to the Chidamber and Kemerer static definition but counting only instance variables that are actually accessed at run time. Mitchell and Power modified the original CK static LCOM to use as a definition of Run-time simple LCOM. They call this R_{LCOM}.

Let $m_1, \ldots m_n$ be the set of methods of a class C. Let $\{I_i^R\}$ be the set of instance variables referenced by m_i at run time. We can define two disjoint sets:

$$PR = \{(I_i^R , I_j^R) \mid I_i^R \cap I_j^R = \emptyset \}$$

$$QR = \{(I_i^R , I_j^R) \mid I_i^R \cap I_j^R \neq \emptyset \},$$

$$\text{then } R_{LCOM} = \frac{|PR|}{|PR|+|QR|} .$$

These are formulated similar to the Chidamber and Kemerer static definitions, in that they employ |P| (number of methods that access the same instance variables) - |Q| (number of methods that do not access the same instance variables). For Simple Dynamic LCOM, P and Q are determined based on which methods do/do not access the same instance variables at runtime.

- Dynamic Call Weighted LCOM by weighting each instance variable by the number of times it is accessed at run time

For Dynamic Call Weighted LCOM, the counts of P and Q occur at runtime, but P and Q are weighted by the number of times that the methods do access the same instance variables at runtime and do not access the same instance variables at runtime

respectively. Note that since the formulation is the same as that of the original Chidamber and Kemerer paper [3], the problems associated with the Chidamber and Kemerer metric formulation as pointed out by Hitz and Montazeri [16] among others, should still be present.

Dufour et al. talk about complexity metrics in their paper; however, they measure it in a way that is different from ours [6]. They use a qualitative approach to complexity whereas we have used a quantitative one. Also their metrics are geared more towards compiler optimization.

Arisholm, Briand and Foyen [1] came up with a definition to precisely measure dynamic coupling based on dynamic analysis of patterns in the code. They were trying to see if there is a correlation between dynamic coupling and change proneness of classes. They concluded that there is significant evidence to show that there exists a correlation between some dynamic coupling measures and change proneness and that their dynamic metrics complement existing static coupling measures.

Hassoun et al. defined a dynamic coupling metric and developed a tool to collect dynamic coupling data represented by message exchanges between objects at runtime. The metric was validated and it was shown to be a good indicator of runtime coupling characteristics of software systems [11][12].

CHAPTER 3

Research Description

In our research, we expand Mitchell and Power and Arisholm et al.'s previous work in runtime cohesion. We also examine runtime complexity, which to our knowledge has not been addressed heretofore.

We present a new suite of Object Oriented Metrics, that we call Dynamic Metrics since they operate at the runtime boundary between static object-oriented metrics and performance metrics. Our suite is different from that of Mitchell and Power [23] and Arisholm et al. [1] in that we specifically address complexity. We create a new dynamic complexity metric that explicitly includes size as part of the measure. There is evidence that size can affect cohesion and we have tried to make this connection explicit. Another dynamic complexity metric that we have created makes use of decision points which are executed at runtime. We have also improved and expanded the cohesion metrics previously suggested by Mitchell and Power and Arisholm et al. To the best of our knowledge, this had not been done before. We derive three new dynamic cohesion metrics which are derived from the LCOM_HS metric.

In order for a suite of metrics to be viable, they must be unique, original and relatively easy to compute. Our hope is that our metrics fulfill these requirements.

3.1 Data Employed

For our research, we examined the Rhino open source software package [25], version 1.6R7. Rhino is an open source implementation of JavaScript written in Java. It can be embedded in Java applications to provide scripting to end users.

Rhino version 1.6R7 has 58,315 lines of code, and 260 classes. The number of classes was determined by examining the source code. The lines of code (LOC) were collected using the metrics tool Understand for Java™ [20][17]. We chose Rhino version 1.6R7 because it was the current version when we began our Dynamic Metrics implementation. Rhino version 1.6R7 was released in 2007.

For our case studies, we ran Rhino [25] using 17 of the standard Rhino test sets that together form the Rhino JavaScript Test suite. The purpose of this test suite is to determine, for each build of Rhino, whether Rhino meets the Rhino specifications. Thus, these test suites have been designed to fully exercise the major portions of Rhino. Each test set contained several files focusing on a particular aspect of Rhino. These test files are JavaScript scripts that act as input data are used to exercise Rhino.

We selected a subset of 69 Rhino classes from the Rhino source code. We chose these 69 classes because they are the core classes of Rhino, the classes needed at a minimum to run Rhino. The other classes are associated classes but are not part of the primary Rhino functionality. We also excluded library objects belonging to java.*, javax.*, sun.*, com.sun.*. We collected our Dynamic Metrics on objects (instantiations) of the Rhino core classes.

In our various case studies, we compared our Dynamic Metrics to three kinds of data: bug data, quality ratings from humans, and static software metrics collected from the source code at compile time. We also in some cases (dynamic cohesion metrics) compare our Dynamic Metrics to earlier Dynamic metrics defined by Mitchell and Power [22].

We derived the bug data by manually mapping bug reports from the Bugzilla repository associated with Rhino version 1.6R7 to the appropriate source code elements that had to be changed to fix the bug. We mapped a subset of reports, a total of 28 bugs for 15 classes. Some reports in the Bugzilla repository are not bug reports, rather they are change requests or issue reports that are not really bug reports; these kinds of reports are not mappable to source code. We mapped actual bug reports that we chose arbitrarily. This kind of activity is extremely labor intensive, mapping a complete set of bugs for even one complete version of a system is a daunting task that is seldom undertaken. Comparing to arbitrarily chosen subsets of bugs is a fairly common practice [26]. When we compared our metrics to bug data, we chose only those classes for which we had available bug data.

The human quality ratings were collected on Rhino version 1.6R1 during a graduate software engineering course (CS552) in spring 2006, as part of collaboration with another researcher (Dr. Patricia Roden). Institutional Review Board (IRB) approval was acquired for the collection of these ratings (APPENDIX A). This Rhino version is very similar to the Rhino version on which we collected the Dynamic Metrics—the core classes are basically unchanged. We compared the classes to make sure this was the case.

We collected Human Data on 40 of the 69 Rhino classes. Only a subset of the classes was examined due to the high level of effort required for the humans to examine the classes. The amount of human "grunt" work required to examine all 69 classes would have been prohibitive. The classes examined were chosen primarily based on size similarity so as to balance the workload between students. Therefore, classes that only defined interfaces, classes that were very small or very large were not selected.

The humans were thirteen students in a graduate level software engineering course. All humans had a bachelor's degree in Computer Science or a related field, and many had industrial experience as well.

They were given an online questionnaire (APPENDIX B), with which they rated the classes. The questionnaire that we used in our research has been used in previous research by Etzkorn [9]. It is also the same questionnaire employed by Roden [27] and Virani [29]. The original Etzkorn questionnaire was paper-based, whereas Roden, Virani, and now we used a web-based version. As part of this earlier work, the internal consistency reliability of this questionnaire was determined using Cronbach's alpha. The reliability of an instrument (questionnaire) examines whether measurements using the instrument are repeatable, and that any randomness in results is a source of measurement error [26]. A value of Cronbach's alpha of greater than 0.70 indicates that the instrument (questionnaire) is reliable. Roden [27] determined that all Cronbach alphas were greater than 0.70 over her data for this instrument (questionnaire). Thus, the instrument can be considered validated.

For each class, the humans employed a 1 through 5 Likert scale to rate the following properties: Cohesion, Coupling, Modularity, Interface, Documentation, Size,

Complexity, Simplicity, Encapsulation, Composition, Inheritance, Abstraction, and Polymorphism. The humans were provided with a definition of each software quality.

3.2 Tools Employed and Data Collection Procedures

Static (compile time) metrics collected on the Rhino version 1.6R7 source code were computed using the tool Understand for Java. In order to collect data at runtime for one of our metrics (AMOC, see below), we used a profiling tool called JProfiler [7]. We wrote a batch file which started the profiling tool and Rhino at the same time. Thus while the various test files were exercising Rhino, JProfiler was collecting runtime data at the same time. JProfiler was set to collect the number of all object instances and the total amount of memory occupied by all these objects of a particular class. Specifically, for AMOC, total memory occupied by an object and total number of instances, of that object were collected. For the AMOC metric, we collected data on all the 69 Rhino core classes [16].

For our other metrics we inserted tags into the source code, which at runtime, resulted in print statements. Upon completion of a run, we counted the resulting printed tags to collect data for our metric. While this may seem like a primitive way to collect data, it is simple, although time consuming. Of course, an improvement in the future would be to create an automated tool to replicate this effort. Every effort was made to ensure that the collection methodology was free from errors. Our original plan was to collect these hand-inserted metrics on all objects associated with the 40 classes for which we possessed Human Data. However, we eliminated 12 classes out of the 40 for these metrics (RuCIVA, RuCCDep and Mod_HS_LCOM1, Mod_HS_LCOM2,

Mod_HS_LCOM3), and collected data only on the objects from 28 classes, because the 12 classes were generating a very large number of objects and consequently an immeasurable number of resulting printed tags. We ran out of memory space on our hard drive twice and also experienced a system crash. Our choice of the 28 classes was somewhat arbitrary—we picked classes that were of reasonable size and similar size; however, the deciding factor was whether or not the code crashed or not. We tried several configurations to try to get the largest possible size that we could run. Note that we collected the data and analyzed it only for our final configuration, thus our final configuration was clearly *not* chosen for how our results were affected.

3.3 Overall Research Approach

All our new Dynamic metrics provide an overall value for all (runtime) objects of a particular class. Thus they take a class-oriented view, similar to the more traditional class-level static metrics. We are not interested in a single object, but rather the group of objects that are instantiations of a particular class.

We defined two new Dynamic Complexity Metrics:

- Runtime Class Complexity based on Decision Points in Code (RuCCDep)
- Average Memory per Object of a Class (AMOC)

These are notable since, to our knowledge, no other researcher has heretofore addressed object-level runtime complexity. We consider complexity from two viewpoints. Our first view is a runtime version of a traditional complexity metric; that is, our RuCCDep is a runtime version of the McCabe's Cyclomatic Complexity metric [18]. We describe it further below. Here we take the view, similar to McCabe's,

that complexity is related to the number of decision points in the code. We calculate complexity based on the number of decision points that actually execute, whereas the original, McCabe's Cyclomatic Complexity calculated complexity based on the number of decision points in the static source code. Our second view is an object-oriented view of a traditional performance metric, memory usage. We focus on the memory uses of objects related to a particular class. The complexity metrics are defined and described below, and the hypotheses that our case studies address are posed.

We also defined four new Dynamic Cohesion Metrics:

- Runtime Cohesion using Instance Variable Accesses (RuCIVA)
- Modified Henderson-Sellers LCOM1 (Mod_HS_LCOM1) - using methods that access attributes at runtime
- Modified Henderson-Sellers LCOM2 (Mod_HS_LCOM2) - using all methods that execute at runtime
- Modified Henderson-Sellers LCOM3 (Mod_HS_LCOM3) – using static no. of methods

Recall that the only previous dynamic cohesion metrics were those defined by Mitchell and Power, which was based on the original Chidamber and Kemerer formulations, which have been shown to have limitations [21][3]. Our work uses the improved Henderson-Sellers formulation [13].

Note that we also provide an additional Dynamic cohesion metric, RuCIVA [19]. This metric is a somewhat different kind of cohesion metric. Here we argue simply that classes whose objects perform a large number of instance variable accesses tend to be less cohesive than classes whose objects perform fewer instance variable accesses. The

argument is size related—classes that do too much work are liable not to be cohesive. A static analogue of this metric would also be possible. Previous work has shown that the results of many of the traditional static object-oriented metrics (such as the Chidamber and Kemerer metrics) vary depending on size [8]. Similarly, it has been shown that cohesion and complexity and cohesion and size are heavily related [28]. This is an attempt to address this issue by formulating metrics that inherently address the issue as to whether size itself is the primary driver of cohesion or complexity.

In our case studies below, we pose various hypotheses. We evaluate the hypotheses by performing correlations of Dynamic metrics with bug data, Dynamic metrics with human ratings, and Dynamic metrics with static metrics. In the case of the cohesion metrics, we also compare our Dynamic Cohesion metrics with the earlier Mitchell and Power dynamic cohesion metric [21].

3.4 Rhino JavaScript Test Suite

We used the Rhino test suite to run on Rhino while we collected runtime data to compute dynamic metrics. Each of the 17 test sets was designed to test a particular aspect of Rhino. These test sets contained individual test cases each of which was run on Rhino. The description of purpose for each set is as given in Table 3.1.

Table 3.1 Rhino Test Set Description of Purpose

Test Set Categories	Brief Description of Test Case	Num. of individual test cases
1	JavaScript Array object and its properties, constructor, length, prototype	28
2	JavaScript Boolean object and its properties, constructor, prototype	22
3	JavaScript Date object, and its properties and methods	149
4	Execution Context – variable instantiation, activation code, global code,	24
5	Expressions- property accessors, new, delete, void, typeof, logical NOT	66
6	Function Objects – Function constructor, prototype, length, arguments	18
7	Global Object – infinity, NaN, unescape, parseInt	16
8	Lexical Conventions – whitespace, line terminators, comments, keywords, future reserved words, identifiers, NULL, numeric, boolean, string literals	75
9	Math Object – LN10, LN2, LOG2E, LOG10E, PI, sqrt, various Math function	39
10	Native Objects – Native ECMAScript Objects	2
11	Number – constructor, prototype, NaN, MAX_VALUE, MIN_VALUE, POSITIVE_INFINITY, NEGATIVE_INFINITY	36
12	ObjectObjects – object prototype, constructor, toString, valueOf	14
13	SourceText – Source Text	2
14	Statements – with, variable, if, while, for, for..in, continue, break	31
15	String – constructor, all its methods	51
16	Type Conversion – ToBoolean, ToNumber, ToString, ToInteger	13
17	Types – undefined type, string type	2

3.5 Human Data

We compared all our Dynamic Metrics to Human Data. Our humans were 13 students in a graduate Software Engineering course. All had a bachelor's degree in Computer Science or a related field.

As mentioned earlier, they were given an online questionnaire with definitions for each of the following properties of Cohesion, Coupling, Modularity, Interface, Documentation, Size, Complexity, Simplicity, Encapsulation, Composition, Inheritance, Abstraction, and Polymorphism. The humans had to rate the classes from 1 to 5 Likert scale on the above mentioned properties. Out of these 13 properties, we used the properties of Cohesion, Coupling and Size as our Human Data in our metric comparisons.

3.6 Bug Data

We collected bug data from the online Bugzilla repository. The Bug Data required hand mapping from Bugzilla bug reports to associated code items. This was extremely labor intensive. It was found that not all bug reports are "mappable." In fact some were found to be feature requests. We mapped 28 bugs chosen from 15 classes, from Rhino Version 1.6R 7.

3.7 Static Complexity Metrics

We compared our Dynamic Complexity metrics to static complexity metrics that we computed using the tool Understand for JAVA. The static metrics we chose to compare our metrics to are SumCyclomatic, MaxCyclomatic and AvgCyclomatic metrics as collected by the tool. AvgCyclomatic, described as Average Cyclomatic Complexity (ACC) in this dissertation, is the average cyclomatic Complexity for all nested functions or methods, MaxCyclomatic described as Maximum Cyclomatic Complexity (MCC) is the maximum cyclomatic complexity of all nested functions or methods whereas

SumCyclomatic described as Sum Cyclomatic Complexity is the sum of cyclomatic complexity of all nested functions or methods.

Prior to performing these correlations, we performed a test for normality on our dynamic metrics data. Our results indicated that the data was not normal; hence we used Spearman's Rank Correlation to perform correlation in our case studies.

3.8 Experimental Procedures

We evaluated our hypotheses using Cohen's [5] correlation magnitude scale:

- <0.1 trivial
- 0.1-0.3 minor
- 0.3-0.5 moderate
- 0.5-0.7 large
- 0.7-0.9 very large
- 0.9-1.0 almost perfect

We used $\alpha = 0.10$ for results to be significant. Thus we are interested in seeing whether any relationship at all exists between runtime cohesion or runtime complexity and more traditional measures of cohesion or complexity.

CHAPTER 4

Dynamic Complexity Metrics

We have defined two new Dynamic Complexity Metrics:

RuCCDep – a runtime metric based on decision points that are executed at runtime

AMOC – a runtime metric based on memory occupied by an object at runtime.

Detailed definitions and explanations are given as follows.

4.1 Runtime Class Complexity using Decision Points (RuCCDeP)

We present a new OO runtime-boundary complexity metric based upon decision points in code [18]. A decision point is a point in code where one option is chosen from more than one available selection. The program execution then follows along the chosen option. It must be noted that the exact path of program execution may be different for each run of the program since a different path may be chosen each time. The decision to choose a path at runtime may be based upon several factors such as user input, conditional values computed at runtime, etc. To the best of our knowledge, we are the first researchers to examine object-oriented (OO) complexity metrics at the OO runtime boundary.

In our implementation of this metric, a subset of 28 classes was selected from Rhino and tags were put at every decision point in each method in these classes. We should point out that we did not tag any methods that did not have any decision points.

Therefore our metric is not *exactly* a runtime boundary version of McCabe's Cyclomatic Complexity (more specifically, it is not a runtime boundary version of the Chidamber and Kemerer WMC metric [3], with complexity for each function calculated using McCabe's Cyclomatic Complexity), in which methods without decision points are assumed to have a decision point of 1. For our metric, if a method does not have a decision point, it is not tagged and therefore not included in the calculation.

For example, an *if* or an *if-else-if* statement can result in totally different code being executed based upon what condition is true at runtime. See Figure 4.1. Given the flowchart, if control follows along the shaded path, we can see that the code that will be executed will be much more complex than the code that would be executed had the control followed any of the non shaded paths because of the presence of 2 additional decision points along the shaded path.

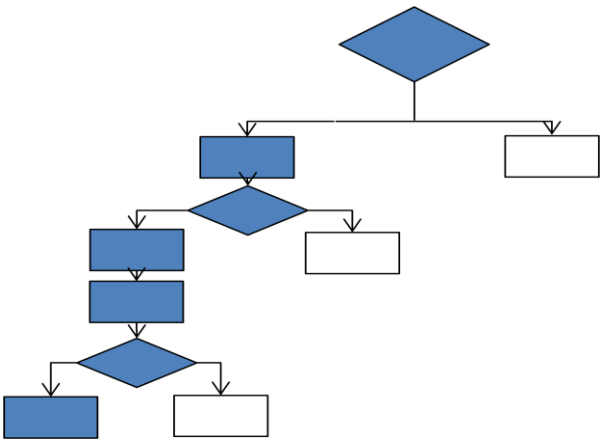

Figure 4.1 Control Flow in Code

The code that is actually executed leads to a change in control flow that alters the complexity of the runtime code. We have used this feature to define our new runtime complexity metric, Runtime Class Complexity using Decision Points (RuCCDep)**.** We

calculated the runtime complexity of objects by counting the number of decision points that were executed at runtime. In order to do this, we inserted print statements in our test code, to tag every decision point. The following schema was followed:

- Every if statement counted as one decision point
- Every if-else statement counted as one decision point
- for, while and do-while loops counted as 1 decision point each; irrespective of the number of iterations. Another potential metric could be formulated to include number of iterations.
- The switch control structure counted as one decision point
- The if-else-if control structures were counted as follows:

```
if (condition)
{
      //if control comes here count as one decision point
          …..
}
else if (condition) ←index (n) is 0
{
     //count as 2 decision points (because if control comes here, it means 2
conditions were tested)
          …..
}
:
:       n times //for nth else-if the number of decision points will be n+2
:
```

```
else
{
….. //the number of decision points will be n+2 because when control comes here
all the above else ifs and the first if statement have been evaluated.
}
```

Try-catch blocks presented an interesting problem. One would be tempted to treat them as an if-else. However this is an overly simplistic view since many lines of code within the try may throw an exception. Indeed, even a single line may have the potential to throw multiple kinds of exceptions. Therefore we considered try-catch beyond the scope of our current research. This is a topic for future research.

We compared our Dynamic Complexity Metrics to Human Data, Bug Data, and static complexity metrics as outlined in the hypotheses.

Runtime Class Complexity based on decision points in code (RuCCDeP) =

$$\frac{total\ no.of\ decision\ points\ for\ an\ object\ +1}{total\ no.of\ objects},$$

where no. of decision points of an object =

$\sum_i^n no.of\ decision\ points\ for\ a\ method,$
where there are n methods with decision points.

Following is a very simple example in pseudo code:

```
class A
{
        function_f1()
        {
                if(i == 1)  //Decision point 1
                {
                        //do something
                }
                while(condition) //Decision point 2
                {
                        //do something
```

```
        }
    }
}
```

At runtime assume 3 objects of type class A were instantiated and function_f1() was called once per instance. Then, since there are two decision points per object

$$RuCCDeP = \frac{2*3+1}{3} = 7$$

Hypothesis #0:

H0: (NULL hypothesis)—There is no significant correlation between the RuCCDep metric and Human Data

H1: (Alternative hypothesis)—There is significant correlation between the RuCCDep metric and Human Data

Hypothesis #1:

H0: (NULL hypothesis)—There is no significant correlation between the RuCCDep metric and Bug Data

H1: (Alternative hypothesis)—There is significant correlation between the RuCCDep metric and Bug Data

Hypothesis #2:

H0: (NULL hypothesis)—There is no significant correlation between the RuCCDep metric and the Maximum Cyclomatic Complexity metric

H1: (Alternative hypothesis)—There is significant correlation between the RuCCDep metric and the Maximum Cyclomatic Complexity metric

Hypothesis #3:

H0: (NULL hypothesis)—There is no significant correlation between the RuCCDep metric and the Sum Cyclomatic Complexity metric

H1: (Alternative hypothesis)—There is significant correlation between the RuCCDep metric and the Sum Cyclomatic Complexity metric

Hypothesis #4:

H0: (NULL hypothesis)—There is no significant correlation between the RuCCDep metric and the Average Cyclomatic Complexity metric

H1: (Alternative hypothesis)—There is significant correlation between the RuCCDep metric and the Average Cyclomatic Complexity metric

4.2 Average Memory per Object of a Class (AMOC)

Memory at runtime is related to complexity, and can be used as an estimate of it. A more complex program is generally assumed to contain more code, and of necessity will require more memory than a less complex program. Note that in this situation it is important to measure memory per object, as opposed to system memory, or memory of the program as a whole. We define AMOC as below [16].

Average Memory per Object of a Class (AMOC) =

$$\frac{\sum_{i=1}^{no.\ of\ object\ instances\ of\ the\ class} no.of\ bytes\ per\ object\ instance}{no.of\ object\ instances\ of\ the\ class}.$$

Hypothesis #5:

H0: (NULL hypothesis)—There is no significant correlation between the AMOC metric and Human Data

H1: (Alternative hypothesis)—There is significant correlation between the AMOC metric and Human Data

Hypothesis #6:

H0: (NULL hypothesis)—There is no significant correlation between the AMOC metric and Bug Data

H1: (Alternative hypothesis)—There is significant correlation between the AMOC metric and Bug Data

Hypothesis #7:

H0: (NULL hypothesis)—There is no significant correlation between the AMOC metric and the Maximum Cyclomatic Complexity metric

H1: (Alternative hypothesis)—There is significant correlation between the AMOC metric and the Maximum Cyclomatic Complexity metric

Hypothesis #8:

H0: (NULL hypothesis)—There is no significant correlation between the AMOC metric and the Sum Cyclomatic Complexity metric

H1: (Alternative hypothesis)—There is significant correlation between the AMOC metric and the Sum Cyclomatic Complexity metric

Hypothesis #9

H0: (NULL hypothesis)—There is no significant correlation between the AMOC metric and the Average Cyclomatic Complexity metric

H1: (Alternative hypothesis)—There is significant correlation between the AMOC metric and the Average Cyclomatic Complexity metric

4.3 Description of Studies

The below table gives a description of how many classes were used for each correlation. For example, the correlation between RuCCDep and Bug Data was performed using 15 classes for which bug data was available. The decision to choose the number of classes for correlation as shown in the table below is explained when the research results are discussed.

Table 4.1 Description of Studies for Dynamic Complexity Metrics

Metric	Study Characteristics	Human Data		Bug Data	MCC	SCC	ACC
		Size	Complexity				
RuCCDep	Normality	No	No	No	No	No	No
	#classes	28	28	15	28	28	28
AMOC	Normality	No	No	No	No	No	No
	#classes	40	40	15	69	69	69

4.4 Research Results

All metrics were calculated on a per test set basis as well as on an overall basis in which we took the average of all 17 test set categories. Except where mentioned, all correlations were performed in two ways, first on a per test set basis and secondly on an overall basis. All results are "significant" at $\alpha = 0.10$. For significant results we list

whether the coefficients are almost perfect, very large, moderate, minor or trivial based on Cohen's correlation magnitude scale [5].

4.4.1 RuCCDep

Hypothesis #0

H0: (NULL hypothesis)—There is no significant correlation between the RuCCDep metric and Human Data

H1: (Alternative hypothesis)—There is significant correlation between the RuCCDep metric and Human Data

We compared RuCCDep with human complexity ratings collected on a subset of the classes. We used a set of 28 classes.

The humans were provided with a definition of complexity, "The complexity of the class relates to simplicity in that the more complex the class, the less simple the class" [3]. The humans evaluated the classes for complexity using a scale that rated: 1 for a simple class, 2 for somewhat complex, 3 for mostly complex, 4 for complex and 5 for very complex.

Inter-rater agreement among the humans was in reasonable ranges over all classes for the human complexity and human size ratings. We calculated inter-rater agreement for this case using rWG [14]. We calculated an Inter-rater Reliability Coefficient of 0.5512 for Complexity and 0.6272 for Size. Therefore, it was reasonable to calculate an average of the human complexity ratings and use that value in comparisons. As mentioned earlier, we compared RuCCDep computed on the above mentioned 28 classes to averages of the ratings of the different human on the same 28 classes.

We calculated the Spearman's Rank Correlation Coefficient in two ways, firstly on a per test set basis and secondly overall taking the average over all test sets.

Table 4.2 Spearman's Correlation Coefficients between RuCCDep and Human Data collected on a per test set basis

Test Sets	Human Complexity Data	
	ρ	p-val
1	0.311	0.224
2	0.217	0.404
3	0.246	0.341
4	0.064	0.807
5	0.283	0.271
6	0.156	0.549
7	0.089	0.735
8	0.245	0.343
9	0.358	0.158
10	0.538	0.026
11	0.238	0.358
12	0.261	0.312
13	0.245	0.343
14	0.216	0.406
15	0.281	0.275
16	0.349	0.170
17	0.245	0.343

When we correlated our metric with the overall Human Complexity data we got a Spearman's Correlation Coefficient of 0.210 and a p-value of 0.417. We consider results to be "significant" at $\alpha = 0.10$. Based upon this, the results on a per test level as well as overall were not significant.

These results are inconclusive. First of all, the argument in favor of a metric such as RuCCDep is quite strong, the number of decision points, as calculated at compile time using the McCabe's Cyclomatic Complexity metric is probably the best understood and best accepted metric in the world—the extension of this concept to runtime is a straightforward argument. Perhaps a human opinion of software based solely on an

examination of the source code is not really a good indication in all cases of software quality. One can argue that the runtime behavior of code can be very difficult for a human to comprehend just by examining static source code. It would seem, however, that the simpler the code, the easier it would be for humans to understand it. Further studies are required.

Based upon the correlation coefficients, we cannot reject the NULL hypothesis.

Hypothesis #1

H0: (NULL hypothesis)—There is no significant correlation between the RuCCDep metric and the bug data

H1: (Alternative hypothesis)—There is significant correlation between the RuCCDep metric and bug data

We calculated Spearman's Rank Correlation between RuCCDep and the bug data that we collected from the Mozilla Bug Database. The correlation coefficient was 0.289 with a p-value of 0.297. At an $\alpha = 0.10$ for "significant," the results are not significant, thus we cannot reject the NULL hypothesis. However we can offer a possible explanation as to the above inconclusive result. Due to the difficulty of manual mapping of bug data to their associated classes, we possessed 28 bugs that map to 15 classes only. We took the RuCCDep metric for those 15 classes only and performed our computation. This is perhaps too small a study, although studies of this fairly small size are typical due to the intensive work required for manual bug mapping [26]. It is possible that we would have arrived at stronger results if we had more bug data. At that point we would be in a much

better position to comment on the results. Bug studies were only done overall, not per test case due to the fairly small number of mapped bugs.

Note, however, that it is impossible to ever claim that a piece of code of any size is 100% free of bugs. The bugs that we don't know about are simply bugs that have not been found yet. Therefore no set of bug data can be deemed to be a completely accurate representation of bugs found in the piece of the software. The bug data we get will always be the subset of the total number of bugs in the software. This problem could potentially skew any case studies employing bug data as a quality measure. However, future studies on larger data sets are clearly necessary.

Based upon the correlation coefficient, we cannot reject the NULL hypothesis.

Hypothesis #2

H0: (NULL hypothesis)—There is no significant correlation between the RuCCDep metric and the Sum Cyclomatic Complexity metric

H1: (Alternative hypothesis)—There is significant correlation between the RuCCDep metric and the Sum Cyclomatic Complexity metric

The results of the per test set comparisons are shown in Table 4.3. As we can see for a majority of test cases, we have a moderate to strong coefficient with a low p-value. If we consider results to be "significant" at $\alpha = 0.10$, we get 16 test cases with significant results. Out of these 8 results we have 8 with moderate coefficients and 8 with large coefficients.

Table 4.3 Spearman's Correlation Coefficient between RuCCDep and SCC on a per test set basis

Test Sets	Sum Cyclomatic Complexity	
	ρ	p-val
1	0.634	0.000
2	0.477	0.010
3	0.490	0.008
4	0.593	0.001
5	0.459	0.014
6	0.405	0.032
7	0.519	0.005
8	0.512	0.005
9	0.540	0.003
10	0.365	0.056
11	0.466	0.012
12	0.485	0.009
13	0.538	0.003
14	0.474	0.011
15	0.549	0.002
16	0.241	0.216
17	0.537	0.003

Secondly we took the average over all test sets and then did an overall correlation with the Sum of Cyclomatic Complexity metric. This gave us strong results, a correlation coefficient of 0.726 and a p-value of < 0.001.

Based on either (or both) of these results, we reject the NULL hypothesis and accept the alternate hypothesis, that RuCCDep is correlated with static Sum of Cyclomatic Complexity metric. However, since it is not a 100% correlation, it is clear that RuCCDep is measuring a somewhat different kind of complexity than did the static complexity metrics, which is what we would have expected.

Hypothesis #3

H0: (NULL hypothesis)—There is no significant correlation between the RuCCDep metric and the Maximum Cyclomatic Complexity metric

H1: (Alternative hypothesis)—There is significant correlation between the RuCCDep metric and the Maximum Cyclomatic Complexity metric

We performed this case study two different two different ways. First, individually for each test case as shown in the table above, we compared RuCCDep with the Maximum Cyclomatic Complexity. The results of the per test set correlations are shown in Table 4.4. As we can see for a majority of test cases, we have a moderate to strong coefficient with a low p-value. If we consider results to be "significant" at $\alpha = 0.10$, we get 16 test cases with significant results. Out of these 16 results, we have 10 with moderate coefficients and 6 with large coefficient.

Table 4.4 Spearman's Correlation Coefficient between RuCCDep and MCC on a per test set basis

Test Sets	Maximum Cyclomatic Complexity	
	ρ	p-val
1	0.615	0.000
2	0.423	0.025
3	0.523	0.004
4	0.507	0.006
5	0.369	0.054
6	0.434	0.021
7	0.493	0.008
8	0.433	0.021
9	0.504	0.006
10	0.336	0.080
11	0.421	0.026
12	0.441	0.019
13	0.459	0.014
14	0.420	0.026
15	0.517	0.005
16	0.112	0.572
17	0.536	0.003

Secondly we took an average over all test sets and then did an overall correlation with the Maximum Cyclomatic Complexity. This gave us moderate to strong results, a correlation coefficient of 0.666 and a p-value of < 0.001.

Based on either (or both) of these results, we reject the NULL hypothesis and accept the alternate hypothesis, that RuCCDep is correlated with Maximum Cyclomatic Complexity metric. However, since it is not a 100% correlation, it is clear that RuCCDep is measuring a somewhat different kind of complexity than did the static complexity metric, which is what we would have expected.

Hypothesis #4

H0: (NULL hypothesis)—There is no significant correlation between the RuCCDep metric and the static Average Complexity Metric

H1: (Alternative hypothesis)—There is significant correlation between the RuCCDep metric and the static Average Complexity Metric

We performed this case study two different ways. First, individually for each test case as shown in the table above, we compared RuCCDep with Average Cyclomatic Complexity. The per test set results are shown in Table 4.6. As we can see for a majority of test sets, we have a very high coefficient with a low p-value. If we consider results to be "significant" at $\alpha = 0.10$, we get 8 test cases with significant results. Out of these 8 results, we have 7 with moderate coefficients and 1 with large coefficients.

Table 4.5 Spearman's Correlation Coefficient between RuCCDep and ACC on a per test set basis

Test Sets	Average Cyclomatic Complexity	
	ρ	p-val
1	0.515	0.005
2	0.278	0.151
3	0.421	0.026
4	0.413	0.029
5	0.291	0.133
6	0.343	0.074
7	0.421	0.026
8	0.291	0.133
9	0.421	0.026
10	0.253	0.194
11	0.280	0.149
12	0.295	0.128
13	0.259	0.184
14	0.293	0.130
15	0.455	0.015
16	-0.036	0.857
17	0.448	0.017

Secondly we averaged the data over all test sets and then did an overall correlation with the Average Cyclomatic Complexity. This gave us fairly strong results though we had expected stronger, based on our earlier case studies. We got a correlation coefficient of 0.420 and a p-value of 0.026.

Based on either (or both) of these results, we reject the NULL hypothesis and accept the alternate hypothesis, that RuCCDep is correlated with the Average Cyclomatic Complexity metric. However, since it is not a 100% correlation, it is clear that RuCCDep is measuring a somewhat different kind of complexity than did the static complexity metrics, which is what we would have expected.

4.4.2 Average Memory per Object of a Class (AMOC)

Hypothesis #5

H0: (NULL hypothesis)—There is no significant correlation between the AMOC metric and Human Data

H1: (Alternative hypothesis)—There is significant correlation between the AMOC metric and Human Data

The humans evaluated the classes on their size and complexity separately. The scale for size was from 1 to 5 with 1 for an extra small class, 2 for a small class, 3 for a medium sized class, 4 for a large class and 5 denoting a very large class. The scale for complexity was 1 for a simple class, 2 for somewhat complex, 3 for mostly complex, 4 for complex and 5 for very complex.

Inter-rater agreement among the humans was in reasonable ranges over all classes for the human complexity and size ratings. We calculated inter-rater agreement for this case using rWG [14]. We calculated an Inter-rater Reliability Coefficient of 0.5512 for Complexity and 0.6272 for Size. Therefore it was reasonable to calculate an average of the human complexity ratings and (separately) the human size ratings, and use those in comparisons. AMOC was correlated to human size and human complexity data. Both correlations were performed on a per test set as shown in Table 4.6, as well as on an overall basis.

Table 4.6 Spearman's Correlation Coefficient between AMOC and Human Data Size and AMOC and Human Data Complexity on a per test set basis

Test Sets	Size		Complexity	
	ρ	p-value	ρ	p-value
1	-0.155	0.480	-0.144	0.512
2	-0.150	0.493	-0.140	0.524
3	-0.136	0.535	-0.115	0.601
4	-0.163	0.457	-0.050	0.821
5	0.030	0.893	0.110	0.618
6	-0.155	0.480	-0.144	0.512
7	-0.115	0.6	-0.119	0.589
8	-0.120	0.586	-0.123	0.576
9	-0.150	0.493	-0.140	0.524
10	-0.089	0.686	0.081	0.714
11	-0.150	0.493	-0.140	0.524
12	-0.155	0.480	-0.144	0.512
13	-0.054	0.805	-0.018	0.937
14	-0.120	0.586	-0.123	0.576
15	-0.155	0.480	-0.144	0.512
16	-0.155	0.480	-0.144	0.512
17	-0.057	0.798	-0.020	0.929

When we correlated AMOC with size on an overall basis, we got a Spearman's Correlation Coefficient of -0.041 and a p-value of 0.852, and with complexity on an overall basis, we got a Spearman's Correlation Coefficient of 0.021 and a p-value of 0.925. If we consider results to be "significant" at $\alpha = 0.10$, we do not get any significant results. The correlation coefficients calculated on a per test basis for all 17 test sets for size and complexity were not significant either. Based upon these coefficients, we cannot reject the NULL hypothesis.

AMOC is measuring the memory used by all objects of a class at runtime, by this argument the more memory you use, the more is "happening," thus the bigger and more complex things are. This does not match the humans' ratings of static source code. Perhaps it is very difficult for humans to understand the actual runtime behavior of code

from simply examining the source code. Normally we would not expect humans to be able to predict the runtime behavior of code just by looking at static source code. Further studies are clearly indicated, especially studies focusing in on what makes the humans consider code as large and complex and other code small and not complex.

Hypothesis #6

H0: (NULL hypothesis)—There is no significant correlation between the AMOC metric and bug data

H1: (Alternative hypothesis)—There is significant correlation between the AMOC metric and bug data

We got a Spearman's Correlation Coefficient of 0.218 and a p-value of 0.435.

If we consider results to be "significant" at $\alpha = 0.10$, the results of the correlation between the AMOC metric and bug data are inconclusive. This could be attributed to the fact that there were not enough bug data to compare to. Based on the p-value of 0.435, we cannot reject the NULL hypothesis.

Hypothesis #7

H0: (NULL hypothesis)—There is no significant correlation between the AMOC metric and the Sum Cyclomatic Complexity metric

H1: (Alternative hypothesis)—There is significant correlation between the AMOC metric and the Sum Cyclomatic Complexity metric

We performed the correlation study two ways: firstly per test case and secondly overall. This correlation was performed with 69 classes since there was AMOC data and static SCC data available for 69 classes. The per test set results are as shown in Table 4.7.

Table 4.7 Spearman's Correlation Coefficient for AMOC and SCC on a per test set basis

Test Sets	Sum Cyclomatic Complexity	
	ρ	p-val
1	0.353	0.003
2	0.338	0.005
3	0.366	0.002
4	0.348	0.004
5	0.363	0.002
6	0..344	0.004
7	0.317	0.009
8	0.364	0.002
9	0.344	0.004
10	0.351	0.004
11	0.336	0.005
12	0.344	0.004
13	0.380	0.001
14	0.342	0.004
15	0.340	0.005
16	0.362	0.002
17	0.357	0.003

If we consider results to be "significant" at $\alpha = 0.10$, all 17 test sets had a significant correlation with 17 moderate coefficients. Overall we got a Spearman's Correlation Coefficient of 0.361 with a p-value of 0.002.

Based upon the correlation coefficients and p-values, we can reject the NULL hypothesis. Again, however, this is not a 100% correlation, which is what we would

have expected—quality measured at runtime is different from quality measured at compile time.

Hypothesis #8

H0: (NULL hypothesis)—There is no significant correlation between the AMOC metric and the Maximum Cyclomatic Complexity metric

H1: (Alternative hypothesis)—There is significant correlation between the AMOC metric and the Maximum Cyclomatic Complexity metric

This correlation was performed with 69 classes since there was AMOC data and static MCC data available for 69 classes. The per test set results are as shown in Table 4.8.

Table 4.8 Spearman's Correlation Coefficient for AMOC and MCC on a per test set basis

Test Sets	Maximum Cyclomatic Complexity	
	ρ	p-val
1	0.258	0.033
2	0.239	0.049
3	0.268	0.027
4	0.266	0.028
5	0.308	0.011
6	0.245	0.044
7	0.226	0.063
8	0.265	0.029
9	0.244	0.045
10	0.275	0.026
11	0.239	0.050
12	0.245	0.044
13	0.291	0.016
14	0.236	0.053

Table 4.8 continued

15	0.244	0.045
16	0.266	0.028
17	0.261	0.032

If we consider results to be "significant" at $\alpha = 0.10$, all 17 test sets had a significant correlation with 16 minor and 1 moderate coefficients. Overall we got a Spearman's Correlation Coefficient of 0.248 with a p-value of 0.041. Based upon these results, we can reject the NULL hypothesis. Again, however, this is not a 100% correlation, which is what we would have expected—quality measured at runtime is different from quality measured at compile time.

Hypothesis #9

H0: (NULL hypothesis)—There is no significant correlation between the AMOC metric and the Average Cyclomatic Complexity metric

H1: (Alternative hypothesis)—There is significant correlation between the AMOC metric and the Average Cyclomatic Complexity metric

This correlation was performed with 69 classes since there was AMOC data and static ACC data available for 69 classes. The per test case results are as shown in Table 4.9.

Table 4.9 Spearman's Correlation Coefficient for AMOC and ACC on a per test set basis

Test Sets	Average Cyclomatic Complexity	
	ρ	p-val
1	0.233	0.056
2	0.211	0.084
3	0.240	0.048
4	0.254	0.037

Table 4.9 continued

5	0.270	0.026
6	0.211	0.085
7	0.219	0.072
8	0.236	0.052
9	0.216	0.077
10	0.214	0.084
11	0.211	0.085
12	0.222	0.069
13	0.210	0.086
14	0.196	0.109
15	0.220	0.071
16	0.241	0.048
17	0.196	0.110

If we consider results to be "significant " at $\alpha = 0.10$, 15 test sets had a "significant" correlation with 15 minor coefficients. Overall we got a Spearman's Correlation Coefficient of 0.193 with a p-value of 0.116. Based upon primarily the per test case correlations, we can reject the NULL hypothesis. Again, however, this is not a 100% correlation, which is what we would have expected—quality measured at runtime is different from quality measured at compile time.

For clarity, and to refer to during our discussion, we summarize our per test set results and our overall results in Tables 4.10 and 4.11.

The following table gives the per test case summary results for the dynamic complexity metrics. We consider results to be "significant" at $\alpha = 0.10$.

Table 4.10 Per Test Set Summary Results for Dynamic Complexity Metrics

Test Set Summary		Humans Size	Humans Complexity	MCC	SCC	ACC
RuCCDep	Significant	N.A	1	16	16	8
	Not Significant	N.A	16	1	1	9
	Minor	N.A	0	0	0	0
	Moderate	N.A	0	10	8	7
	Large	N.A	1	6	8	1
	Very large	N.A	0	0	0	0
AMOC	Significant	0	0	17	17	15
	Not significant	17	17	0	0	2
	Minor	N.A	0	16	0	15
	Moderate	N.A	0	1	17	0
	Large	N.A	0	0	0	0
	Very large	N.A	0	0	0	0

The overall results for dynamic complexity metrics are summarized in the table below.

Table 4.11 Overall Results for Dynamic Complexity Metrics

	Dynamic Metric	Human Data		Bug Data	MCC	SCC	ACC
		Size	Complexity				
RuCCDep	ρ	N.A.	0.210	0.289	0.666	0.726	0.420
	p-value	N.A.	0.417	0.297	<0.001	<0.001	0.026
AMOC	ρ	-0.041	0.021	0.218	0.248	0.361	0.193
	p-value	0.852	0.925	0.435	0.041	0.002	0.116

4.5 Discussion

Our dynamic complexity metrics did not correlate well with Human Data. This could be due to the fact that it is extremely hard for humans to visual the runtime complexity of a piece of software just by looking at the code. What human perceive as complexity at compile time may not exhibit the same behavior at runtime. In a way it is a comparison of unequals.

The Spearman's coefficient between RuCCDep and the static Maximum Cyclomatic complexity shows high positive correlation. Since Maximum Cyclomatic Complexity is an already established metric, a strong correlation indicates that RuCCDep can be used to measure complexity. However, there is not a 100% correlation, which itself indicates that RuCCDep and Maximum Cyclomatic Complexity are not completely the same thing. Since RuCCDep is computed at runtime, we argue it is a better indicator

of dynamic complexity, and thus the "true" complexity of the code than is Maximum Cyclomatic Complexity itself.

Similarly, since Sum Cyclomatic Complexity is an already established metric, a strong correlation indicates that RuCCDep can be used to measure complexity. However, there is not a 100% correlation, which itself indicates that RuCCDep and Sum Cyclomatic Complexity are not completely the same thing. Since RuCCDep is computed at runtime, we argue it is a better indicator of dynamic complexity, and thus the "true" complexity of the code than is Sum Cyclomatic Complexity itself.

It is perhaps reasonable that a metric such as RuCCDep which is a summation of the total number of variable accesses (although divided over the number of objects over which it is calculated) would compare better to max or sum-type complexity values than to an average value.

For the comparisons of AMOC with various versions of McCabe's Cyclomatic Complexity, we got minor correlations since it is not clear that memory usage is directly related to complexity of execution (that is why we proposed the AMOC metric and the RuCCDep metric as separate metrics). One wonders if for certain types of code, however, that increased memory usage might relate to increased complexity of execution, whereas in other types of code it might not. Further studies on this topic would be interesting.

We can say with some certainty that both RuCCDep and AMOC measure dynamic complexity. However we still don't understand how. Further research is required, focusing particularly on what characteristics of code humans think relate to code size and complexity. A larger pool of bug data would also benefit such studies

because bug data is the only true runtime metric that is available for correlation. Any maintenance data such as Mean time Between Failure, Mean time to fix a bug could also be used to correlate with our dynamic complexity metrics and it would be interesting to see what the results would be.

CHAPTER 5

Dynamic Cohesion Metrics

We define 4 new dynamic cohesion metrics, RuCIVA, Mod_HS_LCOM1, Mod_HS_LCOM2 and Mod_HS_LCOM3. The last three metrics are modified versions of the static LCOM-Henderson-Sellers (LCOM_HS) metric. We also calculated Runtime Lack of Cohesion in Methods (R_{LCOM}) as defined by Mitchell and Power under Previous Work in Dynamic Metrics.

5.1 Runtime Cohesion Using Instance Variable Accesses (RuCIVA)

We define a dynamic cohesion metric which we call Runtime Cohesion using Instance Variable Access (RuCIVA) [19]. In order to compute this metric, we inserted print statements in our test code wherever instance variables were accessed by class methods. Batch files were used to run the test code and compute the number of variable accesses.

We calculate RuCIVA by counting the number of instance variable accesses by methods at runtime. Only accesses to instance variables created as part of object creation were considered. Copies of variables passed into functions upon function invocation as pass by value parameters were not considered. Likewise, access to static variables was also not included in this metric (we considered that static variables are not per-object, and we focused on measuring metrics per class/objects).

RuCIVA = $\frac{no.of\ instance\ variable\ accesses\ of\ an\ object\ of\ a\ class}{no.of\ objects\ of\ that\ class}$,

where no. of instance variable access of an object of a class =

$\sum_i^n no.of\ instance\ variable\ accesses\ for\ a\ method,$

where there are n non static methods in the class.

Hypothesis #10:

H0: (NULL hypothesis)—There is no significant correlation between the RuCIVA metric and Human Data

H1: (Alternative hypothesis)—There is significant correlation between the RuCIVA metric and Human Data

Hypothesis #11:

H0: (NULL hypothesis)—There is no significant correlation between the RuCIVA metric and Bug Data

H1: (Alternative hypothesis)—There is significant correlation between the RuCIVA metric and Bug Data

Hypothesis #12:

H0: (NULL hypothesis)—There is no significant correlation between the RuCIVA metric and the R_{LCOM} metric

H1: (Alternative hypothesis)—There is significant correlation between the RuCIVA metric and the R_{LCOM} metric

Hypothesis #13:

H0: (NULL hypothesis)—There is no significant correlation between the RuCIVA metric and the LCOM_HS metric

H1: (Alternative hypothesis)—There is significant correlation between the RuCIVA metric and the LCOM_HS metric

5.2 Modified Henderson-Sellers LCOM1 (Mod_HS_LCOM1)

We modified the LCOM_HS metric as described below to calculate runtime Lack of Cohesion in Methods.

Given a class C with N attributes. Let the total number of objects of a class C at runtime be T. For each attribute n1, let m(n1) be the number of methods accessing n1 at runtime from any object of this class. Let M be the total number of methods of class C that execute and access variables at runtime. Therefore,

M = $\sum_{i=1}^{N} m(n_i)$

Let X = $\frac{1}{T}\left(\frac{M}{N}\right)$

Let Y= X-M

Mod_HS_LCOM1 = $\frac{Y}{1-M}$.

Hypothesis #14:

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM1 metric and Human Data

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM1 metric and Human Data

Hypothesis #15:

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM1 metric and Bug Data

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM1 metric and Bug Data

Hypothesis #16:

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM1 metric and R_{LCOM} metric

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM1 metric and R_{LCOM} metric

Hypothesis #17:

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM1 metric and the LCOM_HS metric

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM1 metric and the LCOM_HS metric

5.3 Modified Henderson-Sellers LCOM2 (Mod_HS_LCOM2)

Given a class C with N attributes. Let the total number of objects of a class C at runtime be T. Let M be the total number of methods of class C that are executed at runtime. This also includes the number of methods that execute and access variables at runtime

Let X = $\frac{1}{T}\left(\frac{M}{N}\right)$

Let Y= X-M

Mod_HS_LCOM2 = $\frac{Y}{1-M}$.

Hypothesis #18

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM2 metric and Human Data

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM2 metric and Human Data

Hypothesis #19

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM2 metric and Bug Data

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM2 metric and Bug Data

Hypothesis #20

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM2 metric and the R_{LCOM} metric

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM2 metric and the R_{LCOM} metric

Hypothesis #21

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM2 metric and the LCOM_HS metric

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM2 metric and the LCOM_HS metric

5.4 Modified Henderson-Sellers LCOM3 (Mod_HS_LCOM3)

Given a class C with N attributes. Let the total number of objects of a class C at runtime be T. Let M be the total static count of methods of class C.

Let X = $\frac{1}{T}\left(\frac{M}{N}\right)$

Let Y= X-M

Mod_HS_LCOM3 = $\frac{Y}{1-M}$.

Hypothesis #22

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM3 metric and Human Data

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM3 metric and Human Data

Hypothesis #23

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM3 metric and Bug Data

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM3 metric and Bug Data

Hypothesis #24

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM3 metric and the R_{LCOM} metric

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM3 metric and the R_{LCOM} metric

Hypothesis #25

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM3 metric and the LCOM_HS metric

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM3 metric and the LCOM_HS metric

For purposes of completion and to compare to our metrics, we also correlated R_{LCOM} with Human Data, Bug Data and LCOM_HS metric.

The hypothesis that we formulated for R_{LCOM} are as listed below.

Hypothesis #26:

H0: (NULL hypothesis)—There is no significant difference between R_{LCOM} and Human Data

H1: (Alternative hypothesis)—There is significant difference between R_{LCOM} and Human Data

Hypothesis #27:

H0: (NULL hypothesis)—There is no significant difference between R_{LCOM} and Bug Data

H1: (Alternative hypothesis)—There is significant difference between R_{LCOM} and Bug Data

Hypothesis #28:

H0: (NULL hypothesis)—There is no significant difference between R_{LCOM} and LCOM_HS metric

H1: (Alternative hypothesis)—There is significant difference between R_{LCOM} and LCOM_HS metric

5.5 Description of Studies

The table below gives a snapshot view of whether the data was normal and the number of classes used for the calculation of the correlation coefficient. For all calculations, 28 classes were used, except for correlations with bugs, where 15 classes were used. This is because we had 15 classes with bug data.

Table 5.1 Description of Studies for Dynamic Cohesion Metrics

Dynamic Cohesion Metric	Study Characteristics	Human Data	Bug Data	R_{LCOM}	LCOM_HS
RuCIVA	Data Normality	No	No	No	No
	#classes	28	15	28	28
Mod_HS_LCOM1	Data Normality	No	No	No	No
	#classes	28	15	28	28
Mod_HS_LCOM2	Data Normality	No	No	No	No
	#classes	28	15	28	28
Mod_HS_LCOM3	Data Normality	No	No	No	No
	#classes	28	15	28	28
R_{LCOM}	Data Normality	No	No	No	No
	#classes	28	15	28	28

5.6 Research Results

All metrics were calculated on a per test set basis as well as on an overall basis in which we took the average of all 17 test set categories. Except where mentioned, all correlations were performed in two ways, first on a per test set basis and second on an overall basis. All results are "significant" at $\alpha = 0.10$. For significant results, we list whether the coefficients are almost perfect, very large, moderate, minor or trivial based on Cohen's correlation magnitude scale [5].

5.6.1 RuCIVA

Hypothesis #10

H0: (NULL hypothesis)—There is no significant correlation between the RuCIVA metric and Human data

H1: (Alternative hypothesis)—There is significant correlation between the RuCIVA metric and Human Data

As discussed earlier human evaluated and rated a subset of 28 classes for cohesion. The human were given the following definition of cohesion: "Cohesion assesses the relatedness of methods and attributes in a class." They rated the classes for cohesiveness as follows: 5 for a class exhibiting excellent cohesiveness, 4 for good, 3 for fair, 2 for poor and 1 for bad. Inter-rater agreement among the humans was in reasonable ranges over all classes for the human cohesion ratings. We calculated inter-rater agreement for this case using rWG [14] to be 0.6848 for Cohesion. Therefore, it was considered reasonable to calculate an average of the human cohesion ratings and use that in comparisons.

Table 5.2 Spearman's Correlation Coefficient between RuCIVA and Human Data on a per test set basis

Test Sets	Human Cohesion Data	
	ρ	p-val
1	-0.162	0.549
2	-0.082	0.763
3	-0.121	0.657
4	-0.195	0.469
5	-0.120	0.657
6	-0.115	0.673
7	-0.091	0.738
8	-0.118	0.665
9	-0.145	0.593
10	-0.177	0.513
11	-0.151	0.578
12	-0.100	0.714
13	0.037	0.891
14	-0.151	0.578
15	-0.103	0.705
16	-0.015	0.955
17	-0.020	0.943

If results are "significant" at $\alpha = 0.10$, then results for all 17 test sets are insignificant. For the overall test set comparison between RuCIVA and human cohesion data, we got a Spearman's Correlation Coefficient of -0.100 and a p-value of 0.714. Based upon these coefficients and p-values, we cannot reject the NULL hypothesis.

Hypothesis #11

H0: (NULL hypothesis)—There is no significant correlation between the RuCIVA metric and Bug Data

H1: (Alternative hypothesis)—There is significant correlation between the RuCIVA metric and Bug Data

When we correlated RuCIVA with bug data we got a Spearman's Correlation Coefficient of 0.264 and a p-value of 0.341. If results are "significant" at $\alpha = 0.10$, then these results are not significant. Based on the high p-value, we are unable to reject the NULL hypothesis. This could be due to the fact that we had a small bug data set.

Hypothesis #12

H0: (NULL hypothesis)—There is no significant correlation between the RuCIVA metric and R_{LCOM} metric

H1: (Alternative hypothesis)—There is significant correlation between the RuCIVA metric and R_{LCOM} metric

R_{LCOM} is a dynamic cohesion metric. We correlated our metric to R_{LCOM}. This would be considered more of an apples to apples comparison since both are dynamic cohesion metrics. At $\alpha = 0.10$, we got a Spearman's correlation coefficient of 0.766 with a p-value < 0.001 for the overall test sets results and large or very large correlation coefficients with low p-values when we did a per test case analysis. Table 5.3 shows the per test case Spearman's Correlation Coefficient. If results are "significant" at $\alpha = 0.10$ then our results are significant for all 17 test sets with 1 moderate, 13 large and 3 very large coefficients.

Table 5.3 Spearman Correlation Coefficient between RuCIVA and R_{LCOM} on a per test set basis

Test Sets	R_{LCOM}	
	ρ	p-val
1	0.677	< 0.001
2	0.664	< 0.001
3	0.662	< 0.001
4	0.733	< 0.001
5	0.693	< 0.001
6	0.461	0.014
7	0.678	< 0.001
8	0.626	< 0.001
9	0.640	< 0.001
10	0.672	< 0.001
11	0.652	< 0.001
12	0.654	< 0.001
13	0.687	< 0.001
14	0.692	< 0.001
15	0.728	< 0.001
16	0.684	< 0.001
17	0.700	< 0.001

Based upon these results, we can reject the NULL hypothesis.

Hypothesis #13

H0: (NULL hypothesis)—There is no significant correlation between the RuCIVA metric and the LCOM_HS metric

H1: (Alternative hypothesis)—There is significant correlation between the RuCIVA metric and the LCOM_HS metric

Table 5.4 Spearman's Correlation Coefficient between RuCIVA and LCOM_HS on a per test set basis

Test Sets	LCOM_HS	
	ρ	p-val
1	0.316	0.151
2	0.351	0.110
3	0.361	0.099
4	0.378	0.083
5	0.401	0.064
6	0.525	0.012
7	0.383	0.078
8	0.331	0.132
9	0.326	0.139
10	0.188	0.402
11	0.334	0.128
12	0.347	0.114
13	0.309	0.162
14	0.350	0.111
15	0.473	0.026
16	0.197	0.381
17	0.310	0.160

If results are "significant" at $\alpha = 0.10$, we get 6 significant results with 1 large and 5 moderate coefficients. As shown in Table 5.4 for test cases 3, 4, 5, 6, 7 and 15, the results were significant at $\alpha = 0.10$. For these test cases, the correlation coefficient was moderate, which gives us some feeling that RuCIVA does measure cohesion. Note that we did not expect RuCIVA to measure exactly the same thing as the static cohesion metric LCOM_HS. This would have meant that RuCIVA had no benefit compared to the static cohesion metric, since LCOM_HS would normally be easier to compute. However, we argue that RuCIVA should measure cohesion more accurately than LCOM_HS, since

RuCIVA is calculated at runtime. What we were examining in this study was whether RuCIVA was actually measuring cohesion.

Overall we got a Spearman's Correlation Cofficient of 0.382 and a p-value of 0.079. Again, the p-value is significant at $\alpha = 0.10$. Since this measures all the code taken together, this gives us more confidence that RuCIVA is indeed measuring cohesion. We thus reject the NULL hypothesis and accept the alternate hypothesis.

5.6.2 Mod_HS_LCOM1

Hypothesis #14

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM1 metric and Human Data

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM1 metric and Human Data

On a per test set basis if results are "significant" at $\alpha = 0.10$, we get 3 results that are significant, with 3 moderate coefficients.

Table 5.5 Spearman's Correlation Coefficient between Mod_HS_LCOM1 and Human Data on a per test set basis

Test Sets	Human Data	
	ρ	p-val
1	0.331	0.211
2	0.452	0.079
3	0.112	0.679
4	0.013	0.963
5	0.350	0.185
6	0.206	0.445
7	0.428	0.098
8	0.438	0.090
9	0.317	0.232
10	0.055	0.839
11	0.070	0.798
12	0.298	0.261
13	0.211	0.434
14	0.336	0.204
15	0.331	0.211
16	0.035	0.898
17	0.331	0.211

When we did an overall correlation with Human Data, we got a Spearman's Correlation Coefficient of 0.175 with a p-value of 0.517.

Based upon the high p-values, we are unable to reject the NULL hypothesis.

Hypothesis #15

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM1 metric and Bug Data

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM1 metric Bug Data

When we did an overall correlation with Human Data, we got a Spearman's Correlation Coefficient of 0.004 with a p-value of 0.989. If results are "significant" at $\alpha = 0.10$, the high p-value indicates the results are insignificant. Thus we are unable to reject the NULL hypothesis.

Hypothesis #16

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM1 metric and R_{LCOM} metric

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM1 metric and R_{LCOM} metric

Mod_HS_LCOM1 was correlated with R_{LCOM} with is also a dynamic cohesion metric. The results are given in Table 5.6. Consider results to be "significant" at $\alpha = 0.10$, we had 16 significant results. Based on Cohen's scale, we had 7 moderate, 5 large and 4 very large correlation coefficients [5].

We also correlated Mod_HS_LCOM1 computed over all test sets to RLCOM. Our results were moderate to strong. Our comparison yielded a Spearman's Correlation Coefficient of 0.598 with a p-value of 0.001. The results are what we expected.

Therefore we can reject the NULL hypothesis. As mentioned earlier, the R_{LCOM} metric is an improvement over the CK-LCOM metric.

Table 5.6 Spearman's Correlation Coefficient between Mod_HS_LCOM1 and R_{LCOM} on a per test set basis

Test Sets	R_{LCOM}	
	ρ	p-val
1	0.354	0.065
2	0.569	0.002
3	0.583	0.001
4	0.404	0.033
5	0.253	0.194
6	0.673	< 0.001
7	0.426	0.024
8	0.578	0.001
9	0.463	0.013
10	0.781	< 0.001
11	0.551	0.002
12	0.455	0.015
13	0.829	< 0.001
14	0.391	0.040
15	0.346	0.071
16	0.767	< 0.001
17	0.709	< 0.001

Since the R_{LCOM} metric is an established metric, the fact that our metric correlates leads us to believe that our metric does measure runtime cohesion. We did expect some difference, however, which is shown since the correlation is not 100%. It is not clear at this time which of these two metrics does a better job of measuring actual cohesion, largely because there is not a well defined validation criteria for cohesion, particularly for cohesion as it affects the runtime behavior of code. Additional and larger studies comparing each of these metrics to bug data, other kinds of maintenance data, etc., are required. However, this is a good first step.

Hypothesis #17

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM1 metric and the LCOM_HS metric

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM1 metric and the LCOM_HS metric

The results computed on a per test basis are as given below. Consider results to be "significant" at $\alpha = 0.10$, we got 3 significant results.

Table 5.7 Spearman's Correlation Coefficient between Mod_HS_LCOM1 and LCOM_HS on a per test set basis

Test Sets	LCOM_HS	
	ρ	p-val
1	-0.472	0.027
2	-0.129	0.567
3	-0.090	0.689
4	-0.126	0.577
5	-0.415	0.055
6	0.015	0.947
7	-0.299	0.176
8	-0.114	0.613
9	-0.274	0.217
10	0.055	0.807
11	-0.155	0.492
12	-0.298	0.178
13	0.266	0.232
14	-0.317	0.151
15	-0.456	0.033
16	0.038	0.867
17	0.091	0.688

When we took the average over all tests, we got a Spearman Correlation Coefficient of 0.262 and a p-value of 0.177, which is not significant. When we performed the correlation on a per test basis, the results were also inconclusive. Therefore overall the

results of the correlation between Mod_HS_LCOM1 and the static LCOM_HS metric were inconclusive.

Therefore, we are unable to reject the NULL hypothesis.

5.6.3 Mod_HS_LCOM2

Hypothesis #18

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM2 metric and Human Data

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM2 metric and Human Data

Consider results to be "significant" at $\alpha = 0.10$, on a per test set basis we do not get any significant results.

Table 5.8 Spearman's Correlation Coefficient between Mod_HS_LCOM2 and Human Data on a per test set basis

Test Sets	Human Data	
	ρ	p-val
1	0.284	0.287
2	0.324	0.221
3	0.422	0.103
4	0.266	0.320
5	0.261	0.329
6	0.370	0.158
7	0.199	0.461
8	0.288	0.279
9	0.275	0.303
10	0.116	0.669
11	0.349	0.186
12	0.279	0.295

Table 5.8 continued

13	0.135	0.619
14	0.275	0.303
15	0.207	0.441
16	0.206	0.444
17	0.284	0.287

Overall we got a Spearman Correlation Coefficient of 0.212 and a p-value of 0.431, which is not significant. Due to the high p-values, the results of the correlation between Mod_HS_LCOM1 and human data were inconclusive. Thus we cannot reject the NULL hypothesis.

Hypothesis #19

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM2 metric and Bug Data

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM2 metric and Bug Data

When we correlated Mod_HS_LCOM2 with bug data we got a Spearman's Correlation Coefficient of -0.256 and a p-value of 0.358. Consider results to be "significant" at $\alpha = 0.10$, the results of the correlation between the Mod_HS_LCOM2 and the bug data are inconclusive. Therefore, we cannot reject the NULL hypothesis.

Hypothesis #20

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM2 metric and R_{LCOM} metric

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM2 metric and R_{LCOM} metric

When we correlated our metric with R_{LCOM} on a per test basis, we got 3 significant results and 13 not significant results with 3 large coefficients at α=0.10. This is puzzling and an analysis to isolate any outlier test cases did not yield any results.

Table 5.9 Spearman's Correlation Coefficient between Mod_HS_LCOM2 and R_{LCOM} on a per test set basis

Test Sets	R_{LCOM}	
	ρ	p-val
1	0.167	0.395
2	0.149	0.450
3	0.191	0.330
4	-0.189	0.335
5	0.013	0.948
6	0.127	0.521
7	0.109	0.580
8	0.198	0.311
9	0.166	0.397
10	0.667	< 0.001
11	0.162	0.411
12	0.160	0.416
13	0.630	< 0.001
14	0.081	0.682
15	0.101	0.608
16	0.243	0.212
17	0.694	< 0.001

When we did an overall correlation of Mod_HS_LCOM2 with R_{LCOM}, we got a Spearman's Correlation Coefficient of 0.552 and a p-value of 0.005. The results of the

correlation are significant at α=0.10. Based on the overall results only, we can reject the NULL hypothesis.

The fact that our metric correlates with this metric gives us confidence to reject the NULL hypothesis on an overall basis and claim that there is significant correlation between the two metrics. Since the R_{LCOM} metric is an established metric, the fact that our metric correlates leads us to believe that our metric does measure runtime cohesion. We did expect some difference, however, which is shown since the correlation is not 100%. It is not clear at this time which of these two metrics does a better job of measuring actual cohesion, largely because there is not a well-defined validation criteria for cohesion, particularly for cohesion as it affects the runtime behavior of code. Additional and larger studies comparing each of these metrics to larger studies of Bug Data, other kinds of maintenance data, etc., are required. However, this is a good first step.

Hypothesis #21

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM2 metric and the LCOM_HS metric

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM2 metric and the LCOM_HS metric

Consider results to be "significant" at α =0.10; we get 10 significant results with 10 moderate coefficients. However, the direction of correlation is negative.

Table 5.10 Spearman's Correlation Coefficient between Mod_HS_LCOM2 and LCOM_HS on a per test set basis

Test Sets	LCOM_HS	
	ρ	p-val
1	-0.341	0.121
2	-0.433	0.044
3	-0.345	0.115
4	-0.445	0.038
5	-0.388	0.075
6	-0.485	0.022
7	-0.379	0.082
8	-0.370	0.090
9	-0.353	0.107
10	0.123	0.585
11	-0.426	0.048
12	-0.343	0.118
13	0.006	0.981
14	-0.407	0.060
15	-0.424	0.049
16	-0.364	0.096
17	0.349	0.111

Overall we got a Spearman's Correlation Coefficient of 0.237 and a p-value of 0.224. Due to the high p-value for overall and somewhat mixed correlations over the individual test cases, we consider the results of the correlation between Mod_HS_LCOM2 and the static LCOM_HS to be inconclusive. Thus we are unable to reject the NULL hypothesis. Further studies and analyses are required.

5.6.4 Mod_HS_LCOM3

Hypothesis #22

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM3 metric and Human Data

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM3 metric and Human Data

Consider results to be "significant" at $\alpha = 0.10$; we got 17 non significant results.

Table 5.11 Spearman's Correlation Coefficient between Mod_HS_LCOM3 and Human Data on a per test set basis

Test Sets	With RLCOM	
	ρ	p-val
1	0.221	0.411
2	0.231	0.389
3	0.240	0.370
4	0.224	0.405
5	0.262	0.326
6	0.208	0.440
7	0.265	0.321
8	0.215	0.424
9	0.215	0.424
10	-0.028	0.917
11	0.188	0.485
12	0.187	0.488
13	0.285	0.285
14	0.261	0.329
15	0.227	0.398
16	0.227	0.398
17	0.221	0.411

When we did an overall correlation, we got Spearman's Correlation Coefficient equal to 0.201 and a p-value of 0.454. Due to the high p-value, the result of the correlation between Mod_HS_LCOM3 and the human data was inconclusive and we cannot reject the NULL hypothesis.

Hypothesis #23

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM3 metric and Bug Data

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM3 metric and Bug Data

We calculated a Spearman's Correlation Coefficient equal to 0.218 and a p-value of 0.435. Considering results to be "significant" at $\alpha = 0.10$, we determined the results to be inconclusive. Therefore, we cannot reject the NULL hypothesis

Hypothesis #24

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM3 metric and R_{LCOM} metric

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM3 metric and R_{LCOM} metric

The per test set correlation coefficients and p-values are given in the Table 5.12. Consider results to be "significant" at $\alpha = 0.10$; we got 2 significant results with 2 moderate coefficients.

Table 5.12 Spearman's Correlation Coefficient between Mod_HS_LCOM3 and R_{LCOM} on a per test set basis

Test Sets	R_{LCOM}	
	ρ	p-val
1	-0.218	0.264
2	-0.127	0.518
3	-0.230	0.239
4	-0.082	0.678
5	-0.196	0.318
6	-0.176	0.369
7	-0.158	0.423
8	-0.144	0.463
9	-0.181	0.356
10	0.389	0.341
11	-0.176	0.371
12	-0.119	0.548
13	0.395	0.037
14	-0.113	0.566
15	-0.265	0.173
16	-0.055	0.781
17	0.489	0.008

When we did an overall correlation, we got a Spearman's Correlation Coefficient of 0.529 and a p-value of 0.885. Due to the high p-value, the results of the correlation between Mod_HS_LCOM3 and the R_{LCOM} metric were inconclusive. Therefore, we are unable to reject the NULL hypothesis.

Hypothesis #25

H0: (NULL hypothesis)—There is no significant correlation between the Mod_HS_LCOM3 metric and the static LCOM_HS

H1: (Alternative hypothesis)—There is significant correlation between the Mod_HS_LCOM3 metric and the static LCOM_HS

The results of the correlation performed on a per test basis are given in Table 5.13.

Consider results to be "significant" at $\alpha = 0.10$; we got 3 significant results with 2 moderate and 1 large coefficient.

Table 5.13 Spearman's Correlation Coefficient between Mod_HS_LCOM3 and LCOM_HS on a per test set basis

Test Sets	Mod_HS_LCOM3	
	ρ	p-val
1	-0.168	0.454
2	-0.018	0.935
3	-0.232	0.299
4	-0.078	0.731
5	-0.240	0.283
6	-0.071	0.755
7	-0.009	0.970
8	-0.076	0.737
9	-0.110	0.627
10	0.448	0.037
11	-0.123	0.586
12	-0.014	0.949
13	0.509	0.016
14	-0.064	0.776
15	-0.244	0.273
16	-0.040	0.861
17	0.480	0.024

When we took the average over all tests, we got a Spearman's Correlation Coefficient of 0.237 and a p-value of 0.224. Due to the high p-value for overall and the inconclusive p-values over the individual test cases, we consider the results of the correlation between Mod_HS_LCOM3 and the static LCOM_HS to be inconclusive. Thus, we are unable to reject the NULL hypothesis. Further studies and analyses are required.

5.6.5 R_{LCOM}

We earlier compared our dynamic cohesion metrics individually to R_{LCOM} which is also a dynamic cohesion metric. R_{LCOM} was defined by Mitchell and Power and is derived from the CK-LCOM metric [23]. For a more complete determination as to how well R_{LCOM} performed compared to our own new dynamic cohesion metrics, we also studied how R_{LCOM} performed in the same kinds of experiments we peformed with our own dynamic cohesion metrics, that is, human data, bugs, and static cohesion metrics. The formulated hypotheses and results of correlation are as given below.

Hypothesis #26

H0: (NULL hypothesis)—There is no significant correlation between the R_{LCOM} metric and Human Data

H1: (Alternative hypothesis)—There is significant correlation between the R_{LCOM} metric and Human Data

We calculated per test case as well as overall Spearman's Correlation Coefficients between R_{LCOM} and Human cohesion data. The results of the per test case analysis are given in the table below.

Consider results to be "significant" at $\alpha = 0.10$; results for 17 test sets were not significant.

Table 5.14 Spearman's Correlation Coefficient between R_{LCOM} and Human Data on a per test set basis

Test Sets	Human Cohesion Data	
	ρ	p-val
1	-0.075	0.809
2	-0.075	0.809
3	-0.048	0.876
4	-0.394	0.183
5	-0.194	0.525
6	-0.031	0.921
7	-0.060	0.846
8	0.039	0.898
9	0.022	0.943
10	-0.012	0.969
11	-0.042	0.891
12	-0.042	0.891
13	0.039	0.898
14	-0.255	0.400
15	-0.042	0.891
16	-0.012	0.969
17	0.039	0.898

When we correlated R_{LCOM} with overall Human data, we got a Spearman's Correlation Coefficient of -0.091 and a p-value of 0.757. Due to a high p-value, the results are insignificant, and we cannot reject the NULL hypothesis.

Hypothesis #27

H0: (NULL hypothesis)—There is no significant correlation between the R_{LCOM} metric and Bug Data

H1: (Alternative hypothesis)—There is significant correlation between the R_{LCOM} metric and Bug Data

The Spearman's Correlation coefficient between RLCOM and bug data was calculated as 0.268 with a p-value of 0.334. Considering results to be "significant" at $\alpha = 0.10$, our

results were inconclusive, and based upon these results, we cannot reject the NULL hypothesis.

Hypothesis #28

H0: (NULL hypothesis)—There is no significant correlation between the R_{LCOM} metric and the LCOM_HS metric.

H1: (Alternative hypothesis)—There is significant correlation between the R_{LCOM} metric and the LCOM_HS metric.

We did a per test case and an overall correlation between R_{LCOM} and static LCOM_HS. The results on a per test case basis are as shown in Table 5.15. Considering results to be "significant" at $\alpha = 0.10$ we got 17 results that are significant with 15 large and 2 very large coefficients.

Table 5.15 Spearman's Correlation Coefficient between R_{LCOM} and LCOM_HS on a per test set basis

Test Sets	LCOM_HS	
	ρ	p-val
1	0.625	0.002
2	0.632	0.002
3	0.646	0.001
4	0.747	< 0.001
5	0.720	< 0.001
6	0.641	< 0.001
7	0.651	0.001
8	0.643	0.001
9	0.654	0.001
10	0.508	0.016
11	0.636	0.001
12	0.618	0.002
13	0.644	0.001
14	0.663	0.001
15	0.618	0.002
16	0.506	0.016
17	0.641	0.001

When we performed an overall correlation between R_{LCOM} and LCOM_HS, we got a Spearman's Correlation coefficient of 0.752 with a p-value < 0.000. Based upon these results, we can reject the NULL hypothesis. This is a very large correlation based on Cohen's magnitude scale [5]. This tends to argue that LCOM_HS should be preferred over R_{LCOM} since it is easier to calculate.

Summaries over all test sets and overall are provided in Tables 5.16 and 5.17, to refer to in our discussion of our overall results.

Table 5.16 Per Test Set Summary Results for Dynamic Cohesion Metrics

Test Set Summary		Humans Cohesion	R_{LCOM}	LCOM_HS
RuCIVA	Significant	0	17	6
	Not Significant	17	0	11
	Minor	0	0	0
	Moderate	0	1	5
	Large	0	13	1
	Very large	0	3	0
Mod_HS_LCOM1	Significant	3	16	3
	Not significant	14	1	14
	Minor	0	0	0
	Moderate	3	7	4
	Large	0	5	0
	Very large	0	4	0
Mod_HS_LCOM2	Significant	0	3	10
	Not Significant	17	14	7
	Minor	0	0	0
	Moderate	0	0	10
	Large	0	3	0

Table 5.16 continued

	Very large	0	0	0
Mod_HS_LCOM3	Significant	0	2	3
	Not significant	17	15	14
	Minor	0		0
	Moderate	0	2	2
	Large	0	0	1
	Very large	0	0	0

Table 5.17 Overall Results for Dynamic Cohesion Metrics

		Human Cohesion	Bug Data	R_{LCOM}	LCOM_HS
RuCIVA	ρ	-0.100	0.264	0.766	0.382
	p-value	0.714	0.341	<0.001	0.079
Mod_HS_LCOM1	ρ	0.175	0.004	0.598	0.262
	p-value	0.517	0.989	0.001	0.177
Mod_HS_LCOM2	ρ	0.212	-0.256	0.552	0.175
	p-value	0.431	0.358	0.005	0.373
Mod_HS_LCOM3	ρ	0.201	0.218	0.529	0.190
	p-value	0.454	0.435	0.885	0.333
R_{LCOM}	ρ	-0.091	0.268	N.A.	0.752
	p-value	0.767	0.334	N.A.	<0.001

5.7 Discussion

As can be seen in Table 5.17, all case studies involving comparisons with Human Data proved to be inconclusive. This could be attributed to the fact that it is very hard for humans to look at static code and predict its behavior at runtime. Therefore, it further strengthens our claim that our metrics are looking at runtime quality attributes, which are very difficult to predict at compile time.

It is also clear that the case studies involving bugs were inconclusive. Larger studies with more bugs are necessary; this is of course difficult due to the heavy manual labor involved in mapping bugs.

RuCIVA correlated very well with R_{LCOM} which gives us confidence that our metric does measure runtime cohesion, because RuCIVA and R_{LCOM} are both dynamic metrics. It did not correlate with LCOM_HS, which indicates that it is measuring something different than a static cohesion metric. Recall that there is not a validation criteria for runtime cohesion, and the degree to which static cohesion and runtime cohesion are the same is unknown—that would be a fruitful area for future research.

The Mod_HS_LCOM1 and Mod_HS_LCOM2 metrics correlated well with R_{LCOM}, which implies that our metrics are indeed measuring runtime cohesion. It is perhaps a bit surprising that they did not correlate with LCOM_HS; however, since R_{LCOM} does correlate with LCOM_HS. However, again LCOM_HS, although shown to be perhaps the best of the various static LCOM implementations, does measure static cohesion and not runtime cohesion [10]. This may indicate our metrics can be more beneficial in measuring runtime cohesion than is R_{LCOM}, since R_{LCOM} apparently measures very similarly to LCOM_HS. It is fairly clear that Mod_HS_LCOM3 is

probably not a useful metric, none of the case studies indicated that this formulation was doing a good job of measuring cohesion.

Overall our metrics had some similarity with R_{LCOM}, but they are not exactly the same. Again, this is hopeful for us. Our metrics are more different from LCOM_HS than R_{LCOM} which means we have more scope for potential usefulness.

CHAPTER 6

Conclusions

Our Dynamic Complexity and Dynamic Cohesion metrics do appear to measure the quality of object –oriented software at runtime. In our research we have seen that Dynamic metrics measure an aspect of code that is not covered by static metrics. We got some good results from our Dynamic Metrics but due to lack of validation standards for dynamic metrics further studies are required. There is no formal procedure that states how dynamic metrics should be compared to static metrics. In particular, it is not known whether human ratings of static code relate at all to runtime behavior of code. Since the field of dynamic metrics is so novel, the lack of pre-set standards make validation extremely difficult. Extensive studies in this area would be interesting.

Bug data are a good way to validate dynamic metrics since they are a runtime measurement, but they are extremely difficult to collect. Larger studies with more bug data are clearly necessary. Also studies related to code maintenance, e.g., Mean Time To Failure, would be useful.

It is extremely hard for humans to look at static code and make predictions about its runtime behavior. If they analyze code and re-analyze the same code later, it is highly unlikely that they will come up with the same ratings. All these factors make validation of dynamic metrics extremely difficult.

We have explored a new territory involving Dynamic Metrics. We feel this gives a good foundation for future research. Our research indicates that Dynamic Metrics do provide a somewhat different picture of the software than Static Metrics. This is a good thing because if they gave the same values as Static Metrics, there would be no benefit to using Dynamic Metrics since they are harder to collect.

The research area using Dynamic Metrics to measure runtime software quality is a very new and emerging field. Our metric definitions are original and unique. At the time of their formulation, we had no idea whether they would be useful or not. Our primary goal in conducting this research was to investigate, formulate and validate dynamic metrics that could be used to analyze software quality at runtime. We got good results; however, a lot remains to be done. We feel we have set the stepping stones for future research in the field of runtime cohesion and complexity metrics. We are encouraged by our results.

CHAPTER 7

Future Research

Dynamic Metrics is still in its infancy. It can be applied to new kinds of software processes like Cloud Computing and Aspect Oriented Software. It would be beneficial (although quite difficult) to establish a validation criteria for comparing Dynamic Metrics to other metrics.

Our metric computations were normalized by dividing by the number of objects. This was also done to keep the numbers comparable to the static metric numbers during correlation coefficient calculations. However, by not considering a per object case, we can potentially define new metrics. Future research is required to determine whether or not this other type of metric would be a better indicator of runtime complexity or runtime cohesion.

As is evident in our study, it is extremely hard for humans to predict how the software will behave at runtime. This would be a fruitful area for future research.

Extensive work in Dynamic metrics would require creating an automated tool to collect the Dynamic Metrics.

APPENDICES

APPENDIX A
IRB Approval Form

June 13, 2005

University of Alabama in Huntsville
Huntsville, AL 35899

Dear Patricia Roden,

As chair of the IRB Human Subjects Committee, I have reviewed your proposal *A Metrics Based Analysis of Software Reusability in Real-time programming Software*, to be carried out during 2005, and have found it meets the necessary criteria for Expedited Review according to 45 CFR 46. I have approved this protocol, and you may commence your research. Please add to the bottom of the informed consent form an expiration date one year from today.

Contact me if you have any questions.

Sincerely,

Dr. Sandra Carpenter,
Acting Chair, UAHIRB

APPENDIX B
Software Quality Questionnaire for Humans

Part I
Directions:
This questionnaire contains statements about code properties. Definition of code properties and criteria is given. Select the appropriate number that you think is most descriptive of the class.

Please enter your unique four digit code:
Please enter the Project class code you are evaluating:

1. **Cohesion**: Assesses the relatedness of methods and attributes in class. Strong overlap in the method parameters and attribute types is an indication of strong cohesion.

How big is the class in terms of number of attributes and number of methods? (A large class is less likely to be cohesive)?

Are methods in the class using disjoint sets of attributes (do there exist methods that have no attributes in common with other methods -- this could be a hint that the class should be broken into two or more classes)?

Are the methods in the class closely related in functionality?

Now, using the criteria stated in the above questions, **rate the class for cohesiveness.**

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

2. Coupling: Defines the interdependency of an object on other objects in a design. It is measure of the number

of other objects that would have to be accessed by an object in order for that object to function correctly.

- Are class methods using global data?
- Does the class have friend functions or classes?

- How many class methods access the attributes of any other class not in the class' s direct hierarchy? (list of direct ancestor classes)

- How many class methods access the methods of any other class not in the class' direct hierarchy? (list of direct ancestor classes)

- How many class methods access any external-to-the-class (standalone) functions (except for methods of other classes)?

- Are there any variable definitions, either in the class definition, or local to a member function, that uses another class as an abstract data type?

Now, using the criteria stated in the above questions, **rate the class for coupling.**

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

3. Modularity: Considering the cohesion and coupling and any other criteria that you think is appropriate , **rate the class for modularity**.

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

4. Interface: A count of number of public methods that are available as services to other classes. This is a measure of the services that a class provides.

- How many public methods are there?

- How many formal parameters, on the average, do the public method definitions have?

- Are the public methods at the appropriate granularity level? That is, do they do too much, not enough, or too little for the functionality provided by the class? Should some of their required functionality be moved to an internal private method, then that method be called by the public method?

- Are the public methods clean or easy to understand?

Now, using the criteria stated in the above questions, and any other criteria you think is appropriate,

rate the class for interface.

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

5. Documentation

- How many comments are there in the class definition?
- How many comments are there in each method, on the average?
- What percentages of methods have any comments at all?
- Are the comments in general well written, understandable, or meaningful?
- Are the identifier names (class names, variable names, method names, etc.) well chosen, understandable, or meaningful?

Now, using the criteria stated in the above questions, and any other criteria you think is appropriate, **rate the class for documentation.**

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

6. Size: This is the measure of how big is the class in terms of number of attributes, methods etc.

- How big is the class in terms of number of attributes?
- How big is the class in terms of number of methods?
- How many lines of executable semicolons are there in the class definition (ignoring comments, blank lines, etc.)?
- How many executable semicolons are there, on the average, in the methods?

- How many formal parameters are there in the method definitions, on the average?

Now, using the criteria stated in the above questions, and any other criteria you think is appropriate, **rate the class for size**.

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

7. Complexity: A measure of the degree of difficulty in understanding and comprehending the internal and external structure of classes and their relationships.

- How complex is the code in the methods, on the average?

Now, using the criteria stated in the above question, and any other criteria you think is appropriate, **rate the class for complexity.**

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

8. Simplicity: Considering the size and complexity and any other criteria that you think is appropriate ,

rate the class for simplicity.

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

9. Encapsulation: Defined as the enclosing of data and behavior within a single construct.

- How many attributes are declared in this class?
- How many attributes are declared private in this class?

Now, using the criteria stated in the above questions, and any other criteria you think is appropriate, **rate the class for encapsulation**

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

10. Composition: This is a measure of aggregation relationships in an object-oriented design.

- How many user-defined classes are employed as data types in the class?

Now, using the criteria stated in the above question, and any other criteria you think is appropriate, **rate the class for composition**

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

11. Inheritance: This is a measure of the "is-a" relationship between classes.

- How many methods are accessible to this class?
- How many methods does this class inherit?

Now, using the criteria stated in the above questions, and any other criteria you think is appropriate, **rate the class for inheritance**

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

12. Abstraction: This is a measure of generalization – specialization aspect of the design.

- How many classes does this class inherit from? (That is the number of classes along all paths from the root classes to this class)

Now, using the criteria stated in the above question, and any other criteria that you think appropriate, **rate the class for abstraction**

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

13. Polymorphism: It is the measure of services that are dynamically determined at run time in an object.

- How many methods exhibit polymorphic behavior?

Now, using the criteria stated in the above question, and any other criteria that you think appropriate, **rate the class for abstraction**

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

Part II

Directions:

Please rate the quality factor for the class

1. Using the criteria stated in the above questions, and any other criteria you think is appropriate**, rate the class for Reusability-in-the-Class:**

5	4	3	2	1
Excellent	Good	Fair	Poor	Bad
○	○	○	○	○

Thank you for your time and effort.

Please submit your answers:

Submit Reset

Definitions provided to Humans collecting data

Cohesion

Cohesion Value: The type of relationship that exists among the elements of each software entity. There are seven types of cohesion listed in an order of increasing desirability.

1. **Coincidental:** No meaningful relationships among the elements of an entity. Difficult to describe the module's function(s). Performs a series of unrelated tasks which seem to be grouped arbitrarily.

2. **Logical**: Entity (at each invocation) performs one of a class of related functions which is selected by the calling module(e.g., "edit all data"). Entity performs more than one function.

3. **Classical or temporal**: Entity performs one of a class of functions that are related in time (program procedure). Entity performs more than one function.

4. **Procedural**: Entity performs more than one function, where the functions are related with respect to the procedure of the problem (Problem procedure).

5. **Communicational**: Entity has procedural strength; in addition, all of the elements "communicate" with one another (e.g. reference same data or pass data among themselves). All functions use the same data.

6. **Informational**: Entity performs multiple functions where the functions (entry points in the unit) deal with a single data structure. Physical packaging together of two or more entities having function strength. All functions use the same data.

7. **Functional**: All entity elements are related to the performance of a single function or accomplishing a single goal. Modules within functional cohesion may be described with a single, simple sentence.

(From Bowen, T.P., Wigle, G.B., and Tsai, J.T. Specification of Software Quality Attributes: Software Quality Evaluation Guidebook, RADC-TR-85-37, Vol. III (of three), Feb. 1985: pp. A-180- A-181 and from Schach, Stephen R. Object-Oriented and Classical Software Engineering Sixty Edition, Boston: McGraw Hill, 2005: pp. 170-174.)

Coupling

Coupling: the type of relationship that exists between two software entities, the degree of interaction between modules, the measure of interdependence of modules or the strength of interconnections. In achieving a highly modular design it is essential to minimize the relationships among software entities. The goal is to design software entities with low coupling. The scale of coupling from worst to best is:

1. **Content Coupling** – One software entity directly references the contents of another software entity. (One module accesses the local data of another)

2. **Common Coupling** – Software entities reference a shared global data structure in place of parameter passing.

3. **External Coupling** – Software entities reference the same externally declared symbol.

4. **Control Coupling** – One software entity passes control elements as arguments to another software entity. This means that one entity controls the logic of the other.

5. **Stamp Coupling** (Data-structured coupling) Two software entities reference the same data structure, which is not global. A data structure which is passed from the calling module is not entirely used by the called module.

6. **Data Coupling** – One software entity calls another and the software entities are not coupled as defined above (in 1 through 5). Each argument passed is either a simple data structure which is entirely used.

(From Bowen, T.P., Wigle, G.B., and Tsai, J.T. Specification of Software Quality Attributes: Software Quality Evaluation Guidebook, RADC-TR-85-37, Vol. III (of three), Feb. 1985: pp. A-181 – A-182 and from Schach, Stephen R. Object-Oriented and Classical Software Engineering Sixty Edition, Boston: McGraw Hill, 2005: pp.176-180).

Modularity

Modularity – Those characteristics of software that provide a structure of highly cohesive components with optimum coupling.

Modularity is the concept of confining specific design decisions or functions into a distinct design element which is as independent of other elements as possible. This independence helps to localize the impact of modifications to within one or a few modules. The advantage of such a modularly designed system is that modules can be replaced or modified without disturbing system functions so long as their interface meets the stated requirements. The primary design goal is to produce a design of the modular structure of the program so that the modules are highly independent. The parts of a module join forces to perform a single specific well-defined function and the data should be explicitly passed as parameters or arguments. In other words we want to maximize the relationships among parts of each module (module strength) and to minimize the relationships among modules (module coupling).

The **Modularity** of software is the key factor of module, function and partial reuse. Since this type of reuse entails interfacing of the existing software with new software, the modularity of the existing software will determine the cost of reusing the software.

(From Presson, P.E., Tsai, J., Bowen, T.P., Post, J.V., and Schmidt, R., Software Interoperability and Reusability, Boeing Aerospace Company, RADC-TR-83-1784m Vol. I, p. 97)

Interface

Definition**:** Interface is measured as the count of number of public methods that are available as service to other classes.

Interface is the measure of the services that a class provides.

(From: J. Bansiya, "A Hierarchical Model For Quality Assessment Of Object-Oriented Designs," *PhD Dissertation,* University of Alabama in, Huntsville, 1997.)

Documentation

Definition: Documentation is the set of attributes of a class, which provide explanation of the functionality and implementation of the class.

(From: L.H Etzkorn, "A Metrics-based Approach to the Automated Identification of Object-Oriented Reusable Software Components," *PhD Dissertation*, University of Alabama in, Huntsville, 1997.)

Size

Definition: The size of the class relates to simplicity in that the larger the class the less simple the class.

(From: L.H Etzkorn, "A Metrics-based Approach to the Automated Identification of Object-Oriented Reusable Software Components," *PhD Dissertation*, University of Alabama in, Huntsville, 1997.)

Complexity

Definition: The complexity of the class relates to simplicity in that the more complex the class, the less simple the class.

(From: L.H Etzkorn, "A Metrics-based Approach to the Automated Identification of Object-Oriented Reusable Software Components," *PhD Dissertation*, University of Alabama in, Huntsville, 1997.)

Simplicity

Definition: Simplicity is attributes of the class that provide for the definition and implementation of the class in the most non-complex and understandable manner.

(From: L.H Etzkorn, "A Metrics-based Approach to the Automated Identification of Object-Oriented Reusable Software Components", *PhD Dissertation*, University of Alabama in, Huntsville, 1997.)

Encapsulation

Definition: Encapsulation in object-oriented design refers to the property of designing the classes that prevent access to attribute declaration by defining them to be private, thus protecting the internal representation of the objects.

(From: J. Bansiya, "A Hierarchical Model For Quality Assessment Of Object-Oriented Designs," *PhD Dissertation*, University of Alabama in, Huntsville, 1997.)

Composition

Definition: Composition measures the "part of","has","consists-of", or "part-whole" relationships, which are aggregation relationships in an object oriented design.

(From: J. Bansiya, "A Hierarchical Model For Quality Assessment Of Object-Oriented Designs," *PhD Dissertation*, University of Alabama in, Huntsville, 1997.)

Inheritance

Definition: Inheritance is the measure of the "is-a" relationship between classes. This relationship is related to the level of nesting of classes in an inheritance hierarchy.

(From: J. Bansiya, "A Hierarchical Model For Quality Assessment Of Object-Oriented Designs," *PhD Dissertation*, University of Alabama in, Huntsville, 1997.)

Abstraction

Definition: Classes in a design that have one or more descendants exhibit this property of abstraction. When designing new systems, high-level design must be abstract enough to accommodate not only current but also all possible future requirements and changes. The abstractions that emerge during design are key in making a design flexible.

(From: J. Bansiya, "A Hierarchical Model For Quality Assessment Of Object-Oriented Designs," *PhD Dissertation*, University of Alabama in, Huntsville, 1997.)

Polymorphism

Definition: When objects in a design have the same base class, it is possible for the sub classes to add or override the implementations of the parent classes and still be able to handle requests addressed to the base class interface. The ability to substitute objects whose interfaces match for one another at run time is called polymorphism.

(From: J. Bansiya, "A Hierarchical Model For Quality Assessment Of Object-Oriented Designs," *PhD Dissertation*, University of Alabama in, Huntsville, 1997.)

REFERENCES

[1] E. Arisholm, "Dynamic Coupling Measures for Object-Oriented Software", Proceedings of the Eighth IEEE Symposium on Software Metrics (METRICS'02).

[2] L. C. Briand, J. W. Daly, and J. K. Wust, "A Unified Framework for Coupling Measurement in Object-Oriented Systems", IEEE Transactions on Software Engineering, January/February 1999.

[3] S. Chidamber and C. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994.

[4] E.S. Cho, C.J. Kim, S.D. Kim, and S.Y. Rhew, "Static and Dynamic Metrics for Effective Object Clustering", APSEC '98 Proceedings of the Fifth Asia Pacific Software Engineering Conference, IEEE Computer Society, Washington DC, USA 1998.

[5] J. Cohen, "Statistical Power Analysis for the Behavioral Sciences", Lawrence Erlbaum Publishing Co., Mahwah, NJ, 1998, 2nd ed.

[6] B. Dufour, K. Driesen, L. Hendren, and C. Verbrugge, "Dynamic Metrics for Java", OOPSLA'03.

[7] http://www.ej-technologies.com/products/jprofiler/overview.html

[8] K. El Emam, S. Benlarbi, N. Goel, W. Melo, H. Lounis, and S. Rai, "The Optimal Class Size for Object-Oriented Software", IEEE Transactions on Software Engineering, Vol. 28, No. 5, May 2002, pp.494-509.

[9] L. H. Etzkorn and C. G. Davis, "Automatically Identifying Reusable OO Legacy Code", IEEE Computer, Vol. 30, No. 10, October 1997, pp. 66-71.

[10] L. H. Etzkorn, S. E. Gholston, J. L. Fortune, C. E. Stein, D. Utley, P.A. Farrington, and G. W. Cox, "A Comparison of Cohesion Metrics for Object-Oriented Systems", Information and Software Technology, Vol. 46, Issue 10, August 2004, pp. 677-687.

[11] Y. Hassoun, S. Counsell, and R. Johnson, "A Dynamic Runtime Coupling Metric for Meta-Level Architectures", Proceedings of the Eighth European Conference on Software Maintenance and Reengineering, 2004.

[12] Y. Hassoun, S. Counsell, and R. Johnson., "Dynamic Coupling Metric: Proof of Concept", IEEE Proceeding-Software, December 2005.

[13] B. Henderson-Sellers, Software Metrics, Prentice-Hall, 1996.

[14] L. R. James, R. G. Demaree, and G. Wolf, "Estimating Within-Group Interrater Reliability With and Without Response Bias", Journal of Applied Psychology, 1984.

[15] K. Keen, R. Mathur, and L. H. Etzkorn**,** "Towards a Measure of Software Intelligence Employing a Runtime Complexity Metric", Proceedings of the IASTED Software Engineering and Applications (SEA) Conference (SEA 2009), Cambridge, MA, Nov. 2-4, 2009.

[16] M. Hitz and B. Montazeri, "Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective", IEEE Transactions on Software Engineering 22 (1996), pp. 267–271.

[17] R. Lincke, J. Lundberg, and W. Löwe, "Comparing Software Metrics Tools", ISSTA'08, July 20–24, 2008, Seattle, WA, USA.

[18] R. Mathur, K. Keen, and L.H. Etzkorn, "Towards an Object-Oriented Complexity Metric at the Runtime Boundary based on Decision Points in Code", ACMSE 2010, April 15-17, 2010, Oxford, MS.

[19] R. Mathur, K. Keen, and L.H. Etzkorn, "Towards a Measure of Object Oriented Runtime Cohesion based on Number of Instance Variable Accesses", ACMSE 2011, March 24-26, 2011, Kennesaw, GA.

[20] http://metrics.sourceforge.net/

[21] A. Mitchell and J.F. Power, "An Empirical Investigation into the Dimensions of Run-time Coupling in Java Programs", Principles and Practice of Programming in Java (PPPJ'04), Las Vegas, NV, June 16-18, 2004, pg 9-14.

[22] A. Mitchell and J.F. Power, "Run-time Cohesion Metrics: An Empirical Investigation", Principles and Practice of Programming in Java (PPJ'04), Las Vegas, NV, June 16-18, 2004, 9-14.

[23] A. Mitchell and J. F. Power, "Towards a Definition of Run-time Object-Oriented Metrics", 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE'2003), Darmstadt University of Technology, Germany, July 21-25, 2003.

[24] A. Mitchell and J.F. Power, "Using Object-Level Run-time Metrics to Study Coupling Between Objects", 2004 International Conference on Software Engineering Research and Practice (SERP'04), Las Vegas, NV, June 21-24, 2004, 532-537.

[25] http:// www.mozilla.org

[26] D. Poshyvanyk, Y.G. Guéhéneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval", IEEE Transactions in Software. Engineering, 33(6), June 2007, pp. 420-432.

[27] P. Roden, "An Examination of Stability and Reusability in Highly Iterative Software", Doctoral Dissertation, The University of Alabama in Huntsville, 2008.

[28] C. Stein, G. W. Cox, L. H. Etzkorn, S. Gholston, S. Virani, P. Farrington, D. Utley, and J. Fortune, "Exploring the Relationship Between Cohesion and Complexity", Journal of Computer Science, 1(2), 2005, pp. 137-144.

[29] S. Virani, "Quantifying Object Oriented Software Quality Factors using Software Metrics", Doctoral Dissertation, The University of Alabama in Huntsville, 2008.