A Proposed Software Maintenance Metric for the Object Oriented Programming Paradigm

by

Cary D. Long

Project Report submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

Approved:

Sallie M. Henry
Committee Chair: Sallie M. Henry

Dennis G. Kafura
Calvin J. Rubbens

October, 1995
Blacksburg, Virginia
A Proposed Software Maintenance Metric for the Object Oriented Programming Paradigm

by

Cary D. Long

Committee Chair: Sallie M. Henry

Computer Science

Abstract

Software complexity metrics have long been used in the procedural programming paradigm. Existing OOP metrics do not address how a class's relation to other classes in the inheritance hierarchy affect it's maintenance needs. This research proposes a new OOP metric, Hierarchy Corrected Metric (HCM) that does address a class's relation to others in the hierarchy. HCM uses the values of the standard OOP metrics for each class plus knowledge of the class inheritance hierarchy to arrive at a revised complexity measurement for each class. That measurement reflects not only internal complexity for the class but also the influence of other classes in the hierarchy to the complexity of the class. Using data from a commercial system HCM was shown to correlate with maintenance better than existing OOP metrics.
Acknowledgments

I would like to thank many people for making it possible for me to complete this work.

Dr. Sallie Henry, my friend, who didn't seem to mind a call from me out of the blue three years since I had last seen her and helping me with every aspect of this work from the ideas and concepts involved to the paperwork at Va. Tech.

Suzanne Costner for bringing me back to life, restoring to me my heart, and lifting the shadow from my soul after a long, long time in the dark.

Frank and Sandy Long, my parents, for always believing in me, and keeping at me to complete this task.

Michelle "Tim" Baldwin, for being a friend and a catalyst to completing this work. "All you've got to do is write it up?"

Chris Stafford for being annoying in the right way when I asked him to.
# Table of Contents

Chapter 1  Software Engineering and Metrics .............................................. 1
    Introduction ......................................................................................... 1
    Some Definitions .................................................................................. 1
    Review of Software Metrics for Procedural Languages ...................... 2
    Summary ............................................................................................. 6

Chapter 2  Object Oriented Programming ...................................................... 7
    Introduction .......................................................................................... 7
    Some definitions ................................................................................... 8
    OOP Metrics ......................................................................................... 12
    Review of existing OOP metrics .......................................................... 12
    Research Direction ................................................................................. 15
    Summary ............................................................................................. 16

Chapter 3  Proposing a new OOP metric ....................................................... 17
    Introduction .......................................................................................... 17
    New metric justification ....................................................................... 17
    Define new metric ................................................................................. 19
    Questions ............................................................................................. 21
    Summary ............................................................................................. 23

Chapter 4  HCM Metric Calculator .............................................................. 24
    Introduction .......................................................................................... 24
    Overview .............................................................................................. 24
    Data Preparation .................................................................................. 25
    HCM Calculator .................................................................................. 26
    Final Analysis ....................................................................................... 27
    Summary ............................................................................................. 29

Chapter 5  Validation of HCM ................................................................. 30
List of Figures

Figure 2.1 Hierarchy Levels ................................................................. 10
Figure 3.2 Influence of Maintenance Needs Example ...................... 18
Figure 3.3 Hierarchy for Sample HCM Calculation ......................... 20
Figure 4.1 Data Flow for HCM Calculation Tool ............................ 24
Figure 5.1 R2 Values for HCM .......................................................... 36
Figure 5.2 Adjusted R2: HCM vs. Base Metric Only ...................... 37
Figure 5.3 WHCM Calculation Example Hierarchy ......................... 39
Figure 5.4 Weighted Ancestor vs. Standard HCM ......................... 41
Figure 5.5 Weighted Descendent vs. Standard HCM ...................... 42
Figure 5.6 Question 2 Example Hierarchy ........................................ 43
List of Tables

Table 5.1 Base Metric Values for each Class in UMIS.............. 31
Table 5.2 Inheritance Heirarchy in UMIS.................................. 32
Table 5.3 HCM Values using Scaling Factor of 0.2.................... 35
Chapter 1  Software Engineering and Metrics

Introduction

Software engineering is the study of the use of objective criteria to monitor and improve both the process of developing software and the software product itself. The objective criteria that software engineers have developed for this study are called metrics. Metrics are some quantitative aspect of the software process (such as man hours devoted to the project) or the software product (such as lines of code). The use of metrics allows software engineers to compare software developed under different systems and at different places and at different times.

Software metrics have been extensively studied in their application to procedural programming languages such as "C" and FORTRAN. But then a new programming paradigm was developed, Object Oriented Programming (OOP). The existing metrics did not address many aspects of the new paradigm. New metrics were developed specifically for OOP. But even these new OOP metrics did not address an important aspect of OOP. This research proposes a new method of revising and improving the existing OOP metrics.

Some Definitions

Definitions of some concepts used in the discussion are included below.
Procedural languages: These languages treat data and processes as completely separate entities. Data types are defined and passed to functions that process the data. There is no connection between a function and the data it operates on. For example a function that was designed solely to perform a particular process on a single type of data (which happens frequently) is not, by any construct of a procedural language, associated to that data.

Software metrics: Software metrics measure quantitative aspects of software. Software metrics are divided into two types: software process metrics and software product metrics. Software process metrics measure items associated with the manner in which an organization develops software, such as man hours devoted to the project. Software product metrics measure the items that are the result of the software process such as the source code and design documents. This research focuses on software product metrics.

Review of Software Metrics for Procedural Languages

It has been generally accepted that the more complex a procedure, function, or entire program is, the more maintenance it requires, and the more difficult that maintenance is to perform. The rational for this idea is that more complex code is more difficult for the writer of the code to understand, and is therefore more likely to contain errors in the first place. Also, in order to maintain a piece of code, one must first understand it, and that more complex code is more difficult to understand. Most metrics for procedural languages therefore attempt to measure complexity. The difference in these metrics is the manner in which they go about measuring how complex a given procedure, function, or program is and what exactly the metrics measure. There are three main types of complexity metrics for procedural programming.
The first type of procedural complexity metrics are lexical measures. Lexical measures base their measurement of complexity on certain lexical tokens in a program. Examples of lexical measure metrics are Halstead's software science complexity metrics [HALM77] and Bail's size metric [BAIW88].

Halstead's software science metrics are based on a program's lexical token counts [HALM77]. Software science metrics are a set of metrics based on the counts of four basic types of lexical tokens in a program. These four basic types of lexical tokens are: number of unique operators, total occurrence of operators, number of unique operands, and total occurrence of operands. The larger the value for one of these metrics, the more complex the program in question is.

Bail reasons that a more complex program is a larger program. Bail defines program size as the number of bits needed to describe an algorithm [BAIW88].

The second type of procedural complexity metrics are graphical analysis measures. These metrics are based on the analysis of patterns in a graph constructed from the control flows in a program. An example of a graphical analysis metric is McCabe's cyclomatic complexity metric [MCCT76] [MCCT89].

McCabe's cyclomatic complexity metric is defined based on the control flows in a procedure or function [MCCT76]. Based on the control flows in the procedure or function, a directed graph is derived. The cyclomatic complexity is based on the complexity of the directed graph. An equivalent and simpler form of the cyclomatic complexity for structural programs is based on the count of simple Boolean conditions in all control constructs in the program. A larger value for a given count implies a more
complex program. McCabe extends cyclomatic complexity to measure structure chart design, to allow measurement to be made at design time [MCCT89].

The third type of procedural complexity metrics measure the interconnection of system components. This interconnection may be based on the statements in a program or the component procedures and functions of the program. There are many interconnection metrics, some examples of which are Adamov's hybrid complexity [ADAR90], Henry-Kafura's information flow complexity [HENS81], McClure's invocation complexity [MCCC78], Robillard's statement inter-connection complexity [ROBP89], and Woodfield's review complexity [WOOS80].

Adamov's hybrid complexity metric is defined for a given program component by three terms: internal complexity (for which Adamov uses McCabe's cyclomatic complexity), positional complexity, and interfacing complexity [ADAR90].

Henry and Kafura define the information flow complexity metric based on the interconnectivity among the different procedures and functions in a program [HENS81]. For a given procedure the information flow complexity metric is: length of procedure X (information flows into the procedure X) information flows out of the procedure)\(^2\). Thus the more information that flows into and out of a procedure, the more complex it becomes.

McClure's invocation complexity definition is based on the possible execution paths in a given program, and the difficulty in tracing the execution path for an arbitrary set of input data [MCCC78]. The invocation complexity is based on the calling hierarchy of a program. The more difficult tracing the execution path becomes, the more difficult maintenance is also.
Robillard's complexity of a program is defined based on the interconnections among statements in that program [ROBP89]. This complexity measurement is based on the information-theory concepts of entropy and excess entropy. The greater the entropy, the more difficult the program is to understand and maintain.

Woodfield defines a metric called review complexity based on the difficulty of understanding the system [WOOS80]. Review complexity is based on the assumption that the more complex a system is, the more time it takes for someone to understand it, and the more time someone must spend reviewing the system before it can be changed.

The above list of complexity measures have been shown to be able to predict the effort required to maintain a program developed in the procedural paradigm. However, these metrics do not address the concepts that OOP adds to programming.
Summary

This chapter gives a brief history of software engineering and the development of software metrics. Also, some of the standard complexity metrics for the procedural paradigm are presented and discussed. In the next chapter, an overview of Object Oriented Programming is presented, and a presentation of the existing metrics for Object Oriented Programming.
Chapter 2  Object Oriented Programming

Introduction

The first programming paradigm extensively studied in software engineering is the procedural paradigm. Software metrics have been extensively studied in their application to procedural programming languages such as "C" and FORTRAN. Several metrics for this paradigm have been proven to have a high correlation with needed maintenance effort. These procedural paradigm metrics were discussed in the last chapter. Recently, there has been a focus on a new type of programming, Object Oriented Programming (OOP). OOP adds several new concepts to a programmer's repertoire, such as classes, objects, inheritance, attributes, methods, and message passing. Special languages for this type of programming have been implemented that include structures that allow the programmer to use these new concepts. Usually these new object oriented languages are a superset of some existing language, such as C++, and Classic ADA\textsuperscript{\texttrademark}. The metrics that were developed for the procedural paradigm do not necessarily function as well with software products developed in the Object Oriented Programming (OOP) paradigm since these new languages introduce an entirely new approach to programming. Several new metrics have been shown to correspond well to maintenance effort required for OOP languages. These new metrics proposed for the OOP paradigm that take into account specifics of the OOP paradigm such as inheritance and message passing correlate well with maintenance effort for OOP programs. However, the current OOP metrics do not take a class's place in relation to others in the inheritance hierarchy into account.
Some definitions

The Object Oriented paradigm introduces several new concepts to programming.

Class: The idea of a class is the basis on which the rest of the Object Oriented paradigm is based. A class is a combination of two types of attributes: data attributes and method attributes. The data attributes hold information about the class and are like variables, the method attributes are a set of operations to be performed on those data attributes. For example, a matrix class would have a data structure to hold the data associated with the matrix, and also methods to perform various matrix related tasks e.g. inverse, transpose, as well as mundane methods to do things like allocate memory for the object, assign values to the positions in the matrix, retrieve such values, and define the size of the matrix.

Object: By itself, a class is merely the definition of a data type, it does not in and of itself make any variables of that type exist in the program, nor does the class definition use any memory in the machine, only variables defined as that type do. A class definition does not in and of itself make any instances of that class appear in the program. An instance of a class in a program is called an object. There may be several objects defined from a given class. It is helpful to think of a class as a type of thing, and an object as a thing of that type. For example based on the matrix class discussed above, there might be three matrices defined, A, B, and C. A, B, and C are all instances of the same type of thing, a matrix.

Attributes: There are two types of attributes of a class, data attributes and method attributes. Data attributes are what the class uses to store data. In the matrix example, the data attributes are the data structure used to store the matrix (most likely an array),
and two size variables, number of rows and number of columns. Method attributes are the operations that can be performed on the data attributes for a class. From the matrix example some of the method attributes are inverse, transpose, allocate, assign, retrieve, and define the size of the matrix.

**Method:** A method is an attribute of a class. It is an operation that can be performed on the data attributes of a class. A method is like a function that is only works on a specific data type, the class the method is defined in. When a class is derived from another class the derived class inherits all the methods that the parent class has (see definition of inheritance below). A class may override a method it inherited by creating a new method of the same name as the one to be replaced.

**Inheritance:** New classes can be derived from classes that are already defined. A derived class has all the data and method attributes from its parent class, but can add new data and method attributes to allow it to function differently. This allows new classes to be developed with little effort. As an example, a square matrix class could be derived from the matrix class discussed above. This new class would add a new method attribute, determinate, and would replace the size definition method with one that only allowed the number of rows and columns to be equal. Note that the programmer now has access to an entirely new class, but only had to implement two new methods, also note that one of the new methods replaces a method that the class had inherited from its parent class.

**Class inheritance hierarchy:** Classes are usually defined in a class hierarchy like a tree. The root of the tree is called a base class that has a few methods that all classes will need defined, such as allocate and deallocate. The base class is at level 0 of the tree. Classes that inherit directly from the base class are level one of the tree, classes that inherit from
level one classes are level two, and so on down the tree as classes are derived from classes on lower levels. See Figure 2.1.

![Figure 2.1 Hierarchy Levels](image)

Some relationships between classes can be defined based on this hierarchy. Given two classes A and B, where B is derived from A, A is called the parent class, and B is called the child class. B's level on the tree is exactly one more than A's. If other classes were derived from A, for example C and D, then B, C, and D would be siblings of each other, since all were derived from the same parent, and all would be at the same level in the tree. Note that each class has exactly one parent, except the base class that has none, but that a class may have any number of children. Some languages allow a class to be derived from more than one parent class. This is called multiple inheritance, but it is not be addressed in this research.

**Encapsulation:** A class provides a well defined interface to it's data through it's methods. An outside caller of the class should never have to directly address the data elements of a class. In fact, some languages allow the programmer to make the data section of a class (and some of the methods) private, meaning that only the class itself may reference them. The reason for this is the users of a class do not need to understand the inner working of
the class in order to use it, they only need to understand the interface to the class provided by the methods of the class. A class that has such an interface is said to be encapsulated. For example, the users of the matrix class discussed above, need never directly address the data structure the matrix is stored in to put a value in the matrix, they simply call the PutAt method. The matrix class could be completely rewritten, for example changing the data structure the matrix values are stored in from a two dimensional array to a list of lists would not effect the user of the class as long as the interface did not change, i.e. PutAt(56,1,1) still put the value 56 at position 1,1 in the matrix. If the matrix class had not been encapsulated and its callers had access to its internal data structures, a change like the one described above would also require the callers to be rewritten. Note that OOP languages allow encapsulation, and some actively support it (via language constructs like private) but it is still up to the designers and programmers to achieve it.

**Message Passing:** Message passing is the term for how objects communicate. This communication is accomplished through the interface defined for the objects. For example if object A wanted the matrix object B to report what value it had at position 1,1, it would pass a message to B to that effect (i.e. it would use B's GetFrom method with the parameters 1,1).

**Runtime binding:** Another unique aspect of OOP is the idea of runtime binding. The programmer can send a message to an object without knowing at the time what type of object it is, the type will be resolved at runtime. Using the matrix example above, the programmer could write a routine to zero all the elements of a matrix. The routine would be passed a matrix object, use the PutAt command, and then return the object. As long as the variable passed into the routine was of class Matrix, any matrix object or *an object of any class with Matrix as an ancestor*, it can be passed to this routine and it will function
correctly. Notice that the programmer only has to develop one routine that works for a class and all its descendants.

The above concepts are the basis of OOP. They allow the programmer to rapidly develop programs based on work already done by using inheritance from already defined classes.

**OOP Metrics**

The metrics defined for the procedural paradigm, as discussed in chapter 1, do not address the new concepts implemented in OOP languages. Metrics have been introduced for OOP languages. These metrics have been shown to correlate well with maintenance effort. However, none of the standard OOP metrics address the place of an object in the object hierarchy, or how that placement can effect needed maintenance effort for that particular object. A new OOP metric type, the Hierarchy Corrected Metric, is proposed, that does take object placement into account. Methods for validating this new metric are then discussed.

**Review of existing OOP metrics**

The metrics described in chapter 1 for the procedural paradigm do not transfer well to the Object Oriented Paradigm. OOP introduces several new concepts such as objects, inheritance, and message passing that the procedural paradigm metrics simply do not address.

Chidamber and Kemerer propose the Method for Object Oriented Software Engineering (MOOSE) that included metrics for object oriented design[CHIS91]. These are Depth in
Tree (DIT), Number of Children (NOC), Response For a Class (RFC), Lack of Cohesion Of Class (LCOM), and Weighted Method per Class (WMC), Number of methods (NOM), and Data Abstraction Coupling (DAC)[CHIS91]. Li provides an analysis of all of these metrics [LIW92].

The DIT metric measures a class’s place in the inheritance hierarchy[CHIS91]. DIT measures the number of classes a given class inherits from (ignoring the case possibility of multiple inheritance which this research does not address). The root class in the inheritance hierarchy has a DIT metric of 0. Chidamber and Kemerer reason that maintenance effort will increase as DIT increases since an understanding of all classes higher in the inheritance hierarchy is necessary for maintenance of the class.

The NOC metric measures the number of direct children of a given class [CHIS91]. Chidamber and Kemerer reason that the larger the NOC metric for a given class, the more classes that are potentially effected by a change to the class, therefore the harder it will be to maintain the given class.

The RFC metric measures the cardinality of the response set of an class [CHIS91]. The response set of a class is defined as the number of local methods plus the number of methods called by local methods. Chidamber and Kemerer reason that the larger the RFC for a given class the more complex the class is, and therefore the harder it will be to maintain.

The LCOM metric measures the lack of cohesion of a class [CHIS91]. LCOM is number of disjoint sets of local methods. Sets do not intersect, and any two methods in a given set both access at least one common local instance variable. Chidamber and Kemerer reason
that the less cohesive a class, the more complex the class and the more difficult it is to maintain.

The WMC metric measures the static complexity of all the methods [CHIS91]. WMC is the summation of the McCabe's cyclomatic complexity for all local methods. Chidamber and Kemerer reason that the more complex the methods, the more complex the class.

The MPC metric measures number of messages sent out from a class [CHIS91]. Chidamber and Kemerer reason that as the number of messages sent out increases, the more dependent a given object is on other objects.

The NOM metric measures number of methods in a class [CHIS91]. Chidamber and Kemerer reason that the more methods a class has, the more complex it is.

The DAC metric measures number of abstract data types in the class [CHIS91]. Chidamber and Kemerer reason that as the number of abstract data types increases, the more dependent a given class is on the classes that define those data types.

Two additional metrics are used in this study. Both attempt to measure the size of the code. Size metrics have been shown to have a high correlation with other metrics in the procedural paradigm.

A suggested size based metric is SIZE_1 and is defined as the number of data attributes in a class plus the number of local methods in a class [KAFD92].
The traditional size metric for procedural languages Lines of code (LOC), is adapted to OOP by defining SIZE_2 for a class as the total lines of source code in that class excluding comments [NANR92].

**Research Direction**

These are the standard metrics used for OOP languages. They have been shown to have a high correlation with maintenance effort. However, the standard OOP metrics do not take a class's place in relation to others in the inheritance hierarchy into account. This research explores the possibility that the position of a class in the inheritance hierarchy effects its maintenance needs. Two related question are addressed:

**Question 1)** Are the maintenance needs of a class influenced by its position in the inheritance hierarchy? It seems possible that the maintenance needs of a class are at some level influenced by the classes it is related to in the inheritance hierarchy. The hypothesis is that yes, the maintenance needs of a class are influenced by its position in the inheritance hierarchy.

**Question 2)** Is a class more heavily influenced by its parent class or its children than by other members of the hierarchy? It also seems likely that the parent or children of a class have more influence than other classes on the maintenance needs of a class. The hypothesis is that yes, the parent and children of a class will more heavily influence the maintenance needs of a class than other members of the hierarchy.
Summary

In this chapter the basic concepts of OOP, class, object, inheritance, attributes, methods, class inheritance hierarchy, encapsulation, and message passing were discussed and defined. Also discussed were the existing OOP metrics, DIT, NOC, CBO, RFC, LCOM, and WMC. These metrics do not address the position of a class in the inheritance hierarchy. Two questions were proposed relating the position of a class in the inheritance hierarchy to maintenance effort.
Chapter 3  Proposing a new OOP metric

Introduction
The OOP metrics described in the previous chapter all have one characteristic in common. They treat each class in the class hierarchy as a stand alone unit, not taking into account each class's unique place in the hierarchy, and the maintenance needs of the other classes in the hierarchy such as parents, children, and siblings, that could effect maintenance in the given class. While DIT (depth in tree) and NOC (number of children) do directly measure an class's place in the hierarchy, they do not measure the interrelationship between an class and the other classes in the hierarchy. MPC (message passing coupling) measures the message traffic out of an class, but does not measure the complexity of the classes that the messages are being sent to.

New metric justification
This hypothesis of this research is that a class's maintenance needs are influenced by the maintenance needs of classes near it in the hierarchy. The reasoning behind this hypothesis is that due to the nature of inheritance changes in one class may necessitate changes in other classes that are related to the first class. This is reflective of the virtual information flow between classes described above.

As an example, consider two very similar classes, A1 and A2. The two have almost identical values for all the standard OOP metrics. A1 is the child of a very simple class, P1, whose standard OOP metrics indicate that it is not likely that P1 will need maintenance. A2 is the child of P2, a very complex class whose standard OOP metrics indicate that it is very likely to need maintenance. Every time that a change needs to be
made to one of the parent classes, it is possible that such change will necessitate a change in the child as well. For example, the parameter list to some method in the parent may need to be changed, necessitating a change in each call to that method in the child. But since P2 is much more likely to need maintenance, A2 is much more likely to need such changes, so is more likely to need maintenance than A1, even though they have identical values for their standard OOP metrics.

This influence of maintenance likelihood is not limited to parent/child relationships in the hierarchy. It should be obvious that in a similar manner to the parent/child relationships described above, any of a node's ancestors could also have a similar effect on its maintenance needs. The same principle holds, however, even when one class is not the descendant of another, due to the manner in which classes are interrelated in the hierarchy.

![Figure 3.2 Influence of Maintenance Needs Example](image)

As an example, consider the case of two classes A and B in a hierarchy where A and B are siblings, i.e. both are direct descendant classes from the same parent class Z (see Figure 3.2 above). Assume that A is a very complex class, and that B is not. More complex classes are more likely to need maintenance. Conventional OOP metrics would measure A and B and make maintenance predictions based on only what can be measured directly
from the code in the classes with A likely and B unlikely to require maintenance. However, maintenance to correct problems in A may effect more than just A. For example, to correct a problem in A, part of the solution may be to add additional parameters to a call to a method in Z, or adding additional data items to Z. Changes like this to Z may require changes to B also. Note that changes were necessary to B only because of problems with A.

**Define new metric**

This research proposes a new metric based on the above ideas, the Hierarchy Corrected Metric (HCM). HCM takes into account not only the complexity of each individual class, but also the complexities of classes that influence the likelihood of maintenance for that class. To calculate the HCM for each class, calculate the conventional OOP metrics as usual. These are called the base metric for that class. Then correct each metric based on the class's place in the hierarchy based on the classes that are near it and the path length to that class. So for any given class, the value of the HCM for a certain base metric b will be:

$$HCM_j = \sum_{i=1}^{n} b_i \cdot s^p$$

where

- $HCM_j$ is the Hierarchy Corrected Metric for that class.
- $n$ is the number of classes in the hierarchy
- $b$ is the base metric value
- $s$ is a scaling factor, and is always between 0 and 1.
- $p$ is the path length between the class i and the given class.

For example, calculate the hierarchy corrected metric for the three classes A, B, and Z above. Below, the value for each class represents the calculated value for some base metric.
As an example, let's calculate the HCM for each class, A, B, and Z shown in the example hierarchy in Figure 3.3. As will be shown in Chapter 4, a reasonable scaling factor is 0.2.

The HCM value for class A, is the sum of three terms, the contribution from class A, from class B, and from class Z. From the HCM definition above, we can calculate A's contribution as $(5 \cdot 0.2^3)$ where 5 is the value of the base metric at A, 0.2 is the scaling factor, and 0 is the path length from A to A. So the value of A's contribution is 5. In a similar manner we can calculate the contribution from Z as $(15 \cdot 0.2^3)$ or 3, and the contribution from B as $(75 \cdot 0.2^2)$ or 3. The final result:

$$HCM_A = (15 \cdot 0.2^3) + (5 \cdot 0.2^0) + (75 \cdot 0.2^2)$$
$$HCM_A = 11$$

In a similar manner HCMs can be calculated for class B and class Z:

$$HCM_B = (15 \cdot 0.2^3) + (5 \cdot 0.2^2) + (75 \cdot 0.2^0)$$
$$HCM_B = 78.2$$
$$HCM_Z = (15 \cdot 0.2^0) + (5 \cdot 0.2^1) + (75 \cdot 0.2^1)$$
$$HCM_Z = 31$$
The metric for each class now reflects the class's place in the hierarchy and the additional maintenance likelihood that exists due to proximity to classes that are complex. Note that for both class A and class Z their HCM value is more than double that for the base metric, due to their having a short path to a relatively complex class. Note also that the value for B, the complex class, changed by a relatively small amount.

Questions

Now that HCM has been defined, it needs to be verified. Recall from the previous chapter the questions posed about a new OOP metric. In Chapter 5 HCM is analyzed and shown to correlate with maintenance effort. The following questions guide that analysis.

Does HCM correlate well with maintenance effort? HCM's definition seems intuitively to address maintenance effort for an class. In the next chapter HCM is analyzed and shown to correlate with maintenance. This is accomplished by using metrics and change data from an actual system. Then, for each HCM metric, perform a regression using the HCM metric as the independent variable and the number of lines changed for that class as the dependent variable. The R squared value generated by the regression measures correlation between the independent and dependent variable.

Can the basic HCM formula be improved? It seems possible that an class's ancestor or descendants could exert more of an influence on its maintenance needs than, for example, its siblings. In the next chapter a modification to the HCM calculation is evaluated that weights the contribution of an ancestor or a descendnet more heavily than
contributions from other classes. Analysis is accomplished in a very similar manner to that described above, with the exception that HCM is calculated in a slightly different manner.
Summary

In this chapter the standard OOP metrics were discussed. The standard metrics do not address an class's interrelation to other classes in the hierarchy. A new metric, Hierarchy Corrected Metric, was defined to address this deficiency in the standard OOP metrics. The analysis of this method of revising existing metrics is presented in a Chapter 5.
Chapter 4  HCM Metric Calculator

Introduction

This chapter describes the tools used to generate the metric values used in the analysis of the HCM metric in the next chapter. A brief overview of the tools used is given, then a more in depth description of each tool follows.

Overview

The calculation and use of the HCM metrics is a three part process (see Figure 4.1). The first part is data preparation, it takes base metrics for each class and the information

![Diagram of the data flow for HCM Calculation Tool](Figure 4.1 Data Flow for HCM Calculation Tool)
necessary to reproduce the class heirarchy as well as change data for each class and produces a data file in a standard format to be used by the actual calculation routine. The second part of the process is calculation of the HCM metric values. The calculation tool takes the data from the standard format file and builds an internal representation of the class heirarchy calculates the HCM metric values for each class and saves the results in a data file that is in standard spreadsheet readable format. This is the section of the tool that is used by all applications. The final part of the process, the analysis section then takes the data from the spreadsheet and performs whatever analysis is necessary on the HCM metrics.

**Data Preparation**

This part of the tool is a front end that takes the base metric data, class change data, and class hierarchy data from whatever format they are in and builds an output file in a format that the HCM calculation tool expects. The output data file is a space delimited ASCII file with the data in the following order:

```plaintext
object name, unique id number, parent id number, DIT, NOC, MPC, RFC, LCM, DAC, WMC, NOM, SIZE_1, SIZE_2, Change for class
```

The parent number of the base class is defined to be -1 to mark its unique status. It may be the case that existing OOP metric tools can be modified to output data in the standard format defined above and eliminate the need for a special tool at this stage in the process.
For this research a data preparation tool was written in "C" that took as its input some of the data files Li [LIW92] generated in his research and converted them to the standard format described above. This is a fairly simple data manipulation tool.

**HCM Calculator**

This is the section of the tool that is constant. It is basically a data pipe that takes its input in the form of the standard data file described above and outputs the HCM metrics for each class within each base metric in standard spreadsheet format.

First the HCM calculator reads in the standard data file and from the information in that file and builds an internal representation of the class hierarchy inheritance tree. Then for each class in the hierarchy a line in the output file is generated. An output data line consists of (in the following order, tab delimited), the name of the class, its unique number in the hierarchy, its parent's unique number, the values for each of the base metrics in the same order listed in the data preparation section above, the change data for this class, and finally 9 entries for each base metric representing varying the scaling factor used in the calculation of HCM from 0.1 to 0.9.

Recall from the previous chapter the definition of the HCM metric:

\[
HCM_j = \sum_{i=1}^{n} b_i \cdot s^p
\]

where

- **HCM\(_j\)** is the Hierarchy Corrected Metric for that class
- \(n\) is the number of classes in the hierarchy
- **b** is the base metric value for class \(i\)
- **s** is a scaling factor, and is always between 0 and 1.
- **p** is the path length between class \(i\) and the given class

26
The function CALC_HCM to calculate HCM for a class given a base metric and a scaling factor is a recursive function that works as follows. Its parameters are current node, scaling factor, last scale, and base metric selector. It returns HCM. The function is first called with current node set to the node of the class that HCM is desired for, scaling factor as desired (between 0 and 1), last scale as 1, and base metric selector as desired. When first called it checks to see if this node has been used before in this calculation (a flag in the internal representation that is initialized to false). The function in pseudo code:

Function CALC_HCM (node, factor, last_scale, base_selector)
if (node.used)
  return 0
else
  node.used = true.
  sum = node.metrics[base_selector]*last_scale
  sum = sum+CALC_HCM(node.parent, factor, (factor*last_scale), base_selector)
  for i = 1 to NumberOfChildren(node)
    sum = sum+CALC_HCM(node.child[i], factor, (factor*last_scale), base_selector)
  return sum

Using the above function the HCM calculation tool is able to produce values for each base metric for each scaling factor. For this research this tool was implemented in "C".

**Final Analysis**

The final analysis section can also vary depending on what the end user wants to analyze. The output from the calculation routine is in a format recognizable by most spreadsheet packages and by SAS. In this research, the final analysis was done with a commercial spreadsheet that included routines for statistical analysis, regression, and could compute $r^2$ values.
Based on the analysis done using varying scaling factors with the HCM calculation tool, it was shown that a scaling factor of 0.2 provides the best correlation for the UMIS data discussed in the next chapter. Intuitively, this value seems plausible. Consider a class, A, where HCM is calculated and a class, B, that has a path length of 1 from class A. If the scaling factor was 0.9, class B would have 90% of the influence that class A's own complexity would have on class A's maintenance needs. This seems too strong, even a factor of 0.5, representing an influence of 50% seems too strong. A figure that seems more reasonable is in the range of 0.1 to 0.25 representing a 10-25% level of influence for classes that have a path length of 1 from one another.
Summary

This chapter described the tool used to calculate the HCM values that are used to validate the HCM metric in the following chapter. The three major sections of the tool, data preparation HCM calculation, and final analysis were discussed. Also, a justification for using a scaling factor of 0.2 in HCM calculations was presented.
Chapter 5 Validation of HCM

Introduction

In this chapter the questions proposed at the end of Chapter 3 are analyzed. The source of the data and the data itself are described. Then each question from Chapter 3 is analyzed and answered.

Description of Data

Software Productivity Solutions, Inc. (SPS) developed an OOP language that is a superset of ADA, Classic ADA\textsuperscript{TM}. SPS also designed and implemented a commercial system, User Interface Management System (UMIS) using Classic ADA\textsuperscript{TM}. Data on the maintenance for UMIS was collected for three years. In that time there were three versions of UMIS released. The maintenance effort for a class in the system is measured as the number of lines changed in the class from the original release. A line change is either an addition or a deletion. If a line is modified, it is counted as both an addition and a deletion.

Li [LIW92] collected the change data for UMIS and also designed and implemented a Classic ADA analyzer to collect the metric values for each object in UMIS as shown in Table 5.1 below. Also shown is the inheritance hierarchy in Table 5.2.
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Change</th>
<th>DIT</th>
<th>NOC</th>
<th>MPC</th>
<th>RFC</th>
<th>LCM</th>
<th>DAC</th>
<th>WMC</th>
<th>NCM</th>
<th>SZ1</th>
<th>SZ2</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>289</td>
<td>0</td>
<td>6</td>
<td>9</td>
<td>81</td>
<td>27</td>
<td>7</td>
<td>59</td>
<td>37</td>
<td>44</td>
<td>71</td>
</tr>
<tr>
<td>window</td>
<td>192</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>57</td>
<td>20</td>
<td>13</td>
<td>41</td>
<td>32</td>
<td>45</td>
<td>419</td>
</tr>
<tr>
<td>popup_window</td>
<td>26</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>29</td>
<td>6</td>
<td>2</td>
<td>9</td>
<td>11</td>
<td>13</td>
<td>86</td>
</tr>
<tr>
<td>select</td>
<td>34</td>
<td>3</td>
<td>0</td>
<td>9</td>
<td>25</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>10</td>
<td>14</td>
<td>131</td>
</tr>
<tr>
<td>popup_menu</td>
<td>18</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>9</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>dialog</td>
<td>16</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>12</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>26</td>
</tr>
<tr>
<td>name_dialog</td>
<td>15</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>10</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>61</td>
</tr>
<tr>
<td>alert</td>
<td>18</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>20</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>94</td>
</tr>
<tr>
<td>screen</td>
<td>93</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>46</td>
<td>13</td>
<td>5</td>
<td>37</td>
<td>20</td>
<td>26</td>
<td>296</td>
</tr>
<tr>
<td>graphic</td>
<td>205</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>67</td>
<td>6</td>
<td>21</td>
<td>23</td>
<td>40</td>
<td>61</td>
<td>194</td>
</tr>
<tr>
<td>string80_rendering</td>
<td>168</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>40</td>
<td>13</td>
<td>6</td>
<td>44</td>
<td>20</td>
<td>26</td>
<td>316</td>
</tr>
<tr>
<td>yn_boolean_rendering</td>
<td>20</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>17</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>76</td>
</tr>
<tr>
<td>it_boolean_rendering</td>
<td>30</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>17</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>76</td>
</tr>
<tr>
<td>integer_rendering</td>
<td>12</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>9</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>33</td>
</tr>
<tr>
<td>float_rendering</td>
<td>48</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>17</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>12</td>
<td>15</td>
<td>74</td>
</tr>
<tr>
<td>menu_title</td>
<td>39</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>29</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>13</td>
<td>16</td>
<td>90</td>
</tr>
<tr>
<td>enumeration_rendering</td>
<td>16</td>
<td>3</td>
<td>0</td>
<td>9</td>
<td>24</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>81</td>
</tr>
<tr>
<td>view</td>
<td>253</td>
<td>2</td>
<td>3</td>
<td>12</td>
<td>101</td>
<td>31</td>
<td>5</td>
<td>69</td>
<td>39</td>
<td>44</td>
<td>439</td>
</tr>
<tr>
<td>vertical_view</td>
<td>30</td>
<td>3</td>
<td>0</td>
<td>9</td>
<td>30</td>
<td>9</td>
<td>0</td>
<td>16</td>
<td>9</td>
<td>9</td>
<td>142</td>
</tr>
<tr>
<td>indented_list</td>
<td>30</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>12</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>79</td>
</tr>
<tr>
<td>horizontal_view</td>
<td>30</td>
<td>3</td>
<td>0</td>
<td>9</td>
<td>30</td>
<td>9</td>
<td>0</td>
<td>16</td>
<td>9</td>
<td>9</td>
<td>143</td>
</tr>
<tr>
<td>rectangle</td>
<td>48</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>11</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>45</td>
</tr>
<tr>
<td>line</td>
<td>50</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>17</td>
<td>9</td>
<td>2</td>
<td>12</td>
<td>9</td>
<td>11</td>
<td>83</td>
</tr>
<tr>
<td>data</td>
<td>16</td>
<td>1</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>tree</td>
<td>17</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>64</td>
<td>6</td>
<td>2</td>
<td>45</td>
<td>32</td>
<td>34</td>
<td>338</td>
</tr>
<tr>
<td>string80_data</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>8</td>
<td>27</td>
</tr>
<tr>
<td>ratio</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>15</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>11</td>
<td>44</td>
</tr>
<tr>
<td>offset_ratio</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>34</td>
</tr>
<tr>
<td>list_data</td>
<td>26</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>46</td>
<td>26</td>
<td>2</td>
<td>30</td>
<td>31</td>
<td>33</td>
<td>263</td>
</tr>
<tr>
<td>integer_data</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>26</td>
</tr>
<tr>
<td>float_data</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>26</td>
</tr>
<tr>
<td>dictionary</td>
<td>18</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>7</td>
<td>1</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>59</td>
</tr>
<tr>
<td>boolean_data</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>12</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>26</td>
</tr>
<tr>
<td>context</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>21</td>
<td>12</td>
<td>8</td>
<td>0</td>
<td>18</td>
<td>26</td>
<td>60</td>
</tr>
<tr>
<td>action</td>
<td>14</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>14</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>37</td>
</tr>
<tr>
<td>quit</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>done</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>cancel</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Uhrs</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>32</td>
<td>7</td>
<td>2</td>
<td>10</td>
<td>14</td>
<td>18</td>
<td>84</td>
</tr>
<tr>
<td>Class Name</td>
<td>ID NUMBER</td>
<td>Parent's ID #</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>---------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>object</td>
<td>0</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>window</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>popup_window</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>selector</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>popup_menu</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dialog</td>
<td>5</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>name_dialog</td>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>alert</td>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>screen</td>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>graphic</td>
<td>9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>string80_rendering</td>
<td>10</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>yn_boolean_rendering</td>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tf_boolean_rendering</td>
<td>12</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>integer_rendering</td>
<td>13</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float_rendering</td>
<td>14</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>menu_title</td>
<td>15</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>enumeration_render</td>
<td>16</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>view</td>
<td>17</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vertical_view</td>
<td>18</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>indented_list</td>
<td>19</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>horizontal_view</td>
<td>20</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rectangle</td>
<td>21</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>line</td>
<td>22</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>23</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tree</td>
<td>24</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>string80_data</td>
<td>25</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ratio</td>
<td>26</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>offset_ratio</td>
<td>27</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>list_data</td>
<td>28</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>integer_data</td>
<td>29</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float_data</td>
<td>30</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dictionary</td>
<td>31</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boolean_data</td>
<td>32</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>context</td>
<td>33</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>action</td>
<td>34</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>quit</td>
<td>35</td>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>done</td>
<td>36</td>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cancel</td>
<td>37</td>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>umirs</td>
<td>38</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**A quick review of some Statistical terms**

An understanding of the following terms from statistical analysis is necessary for the discussions that follow.

**dependent variable:** A random variable to be predicted.

**independent variable:** A random variable that is used to predict the dependent variable.

**sample:** A set of classes drawn from the commercial environment, a subset of the population.

**population:** the set of all classes existing in a commercial environment.

**R-square:** the percentage of the variance in the dependent variable accounted for by the independent variables in a regression model in the sample, also known as the coefficient of determination, this value is a measure of how close the regression line fits the data.

**Adjusted R-square:** the percentage of the variance in the dependent variable accounted for by the independent variables in a regression model in the population.

**Review Question 1**

Does HCM correlate well with maintenance? Given the definition of HCM from Chapter 3 show that HCM metrics calculated from the UMIS data do correlate well with the change data for UMIS.
Procedure

Using the data in Table 5.1 and knowledge of the object hierarchy from Table 5.2 calculate the HCM for each object in the UMIS system for each different base metric. This data is shown in Table 5.3 below. For each HCM metric, perform a regression using the HCM metric as the independent variable and the number of lines changed for that object as the dependent variable. The $r^2$ value generated by the regression measures correlation between the independent and dependent variable.
<table>
<thead>
<tr>
<th>Class Name</th>
<th>DI</th>
<th>NCC</th>
<th>MPC</th>
<th>RFC</th>
<th>LCOM</th>
<th>DAC</th>
<th>WMC</th>
<th>NCM</th>
<th>S21</th>
<th>S22</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>2.82</td>
<td>9.77</td>
<td>18.81</td>
<td>140.6</td>
<td>44.61</td>
<td>17.95</td>
<td>89.9</td>
<td>69.68</td>
<td>88.04</td>
<td>356</td>
</tr>
<tr>
<td>window</td>
<td>2.4</td>
<td>3.72</td>
<td>15.28</td>
<td>91.01</td>
<td>30.19</td>
<td>16.68</td>
<td>59.57</td>
<td>47.84</td>
<td>64.66</td>
<td>501.1</td>
</tr>
<tr>
<td>popup_window</td>
<td>4.86</td>
<td>4.78</td>
<td>12.62</td>
<td>59.1</td>
<td>16.37</td>
<td>6.41</td>
<td>23.05</td>
<td>25.46</td>
<td>32.17</td>
<td>238.3</td>
</tr>
<tr>
<td>selector</td>
<td>3.85</td>
<td>0.96</td>
<td>11.16</td>
<td>35.82</td>
<td>9.99</td>
<td>4.16</td>
<td>12.29</td>
<td>14.69</td>
<td>19.87</td>
<td>173.4</td>
</tr>
<tr>
<td>popup_menu</td>
<td>3.85</td>
<td>0.96</td>
<td>4.44</td>
<td>20.46</td>
<td>7.11</td>
<td>1.28</td>
<td>4.61</td>
<td>8.93</td>
<td>10.27</td>
<td>72.62</td>
</tr>
<tr>
<td>dialog</td>
<td>4.62</td>
<td>1.92</td>
<td>5.21</td>
<td>25.26</td>
<td>9.8</td>
<td>2.24</td>
<td>4.61</td>
<td>11.62</td>
<td>14.31</td>
<td>84.33</td>
</tr>
<tr>
<td>name_dialog</td>
<td>4.76</td>
<td>0.38</td>
<td>4.88</td>
<td>14.65</td>
<td>5.8</td>
<td>0.45</td>
<td>0.92</td>
<td>6.16</td>
<td>8.62</td>
<td>75.43</td>
</tr>
<tr>
<td>alert</td>
<td>3.85</td>
<td>0.96</td>
<td>8.28</td>
<td>31.02</td>
<td>9.03</td>
<td>3.2</td>
<td>9.41</td>
<td>11.81</td>
<td>15.07</td>
<td>137.9</td>
</tr>
<tr>
<td>screen</td>
<td>1.52</td>
<td>1.95</td>
<td>14.32</td>
<td>72.27</td>
<td>21.4</td>
<td>8.39</td>
<td>53.5</td>
<td>33.14</td>
<td>42.57</td>
<td>355.4</td>
</tr>
<tr>
<td>graphic</td>
<td>4.1</td>
<td>7.52</td>
<td>19.93</td>
<td>132</td>
<td>28.08</td>
<td>26.52</td>
<td>68.21</td>
<td>69.08</td>
<td>95.68</td>
<td>457.5</td>
</tr>
<tr>
<td>string80_rendering</td>
<td>6.2</td>
<td>7.26</td>
<td>15.31</td>
<td>86.49</td>
<td>24.82</td>
<td>12.41</td>
<td>63.37</td>
<td>42.04</td>
<td>54.46</td>
<td>477.4</td>
</tr>
<tr>
<td>yn_boolean_rendering</td>
<td>4.12</td>
<td>1.45</td>
<td>8.62</td>
<td>33.62</td>
<td>10.72</td>
<td>2.48</td>
<td>22.27</td>
<td>14.17</td>
<td>16.65</td>
<td>168.4</td>
</tr>
<tr>
<td>tz_boolean_rendering</td>
<td>4.12</td>
<td>1.45</td>
<td>8.82</td>
<td>33.62</td>
<td>10.72</td>
<td>2.48</td>
<td>22.27</td>
<td>14.17</td>
<td>16.65</td>
<td>168.4</td>
</tr>
<tr>
<td>integer_rendering</td>
<td>4.12</td>
<td>1.45</td>
<td>5.94</td>
<td>25.94</td>
<td>9.76</td>
<td>2.48</td>
<td>15.55</td>
<td>13.21</td>
<td>15.69</td>
<td>127.2</td>
</tr>
<tr>
<td>float_rendering</td>
<td>4.12</td>
<td>1.45</td>
<td>5.94</td>
<td>33.62</td>
<td>10.72</td>
<td>5.36</td>
<td>15.55</td>
<td>19.93</td>
<td>25.29</td>
<td>166.5</td>
</tr>
<tr>
<td>menu_title</td>
<td>4.12</td>
<td>1.45</td>
<td>9.78</td>
<td>45.14</td>
<td>11.68</td>
<td>5.36</td>
<td>20.35</td>
<td>20.89</td>
<td>26.25</td>
<td>181.9</td>
</tr>
<tr>
<td>enumeration_rendering</td>
<td>4.12</td>
<td>1.45</td>
<td>11.7</td>
<td>40.34</td>
<td>9.76</td>
<td>3.44</td>
<td>17.47</td>
<td>13.21</td>
<td>16.65</td>
<td>173.2</td>
</tr>
<tr>
<td>view</td>
<td>4.47</td>
<td>4.38</td>
<td>19.73</td>
<td>137.2</td>
<td>39.98</td>
<td>10.1</td>
<td>86.41</td>
<td>55.86</td>
<td>65.98</td>
<td>582.8</td>
</tr>
<tr>
<td>vertical_view</td>
<td>3.77</td>
<td>0.88</td>
<td>12.59</td>
<td>56.24</td>
<td>16.64</td>
<td>2.02</td>
<td>32.64</td>
<td>19.81</td>
<td>21.84</td>
<td>252.9</td>
</tr>
<tr>
<td>indented_list</td>
<td>3.77</td>
<td>0.88</td>
<td>7.79</td>
<td>38.96</td>
<td>13.76</td>
<td>2.02</td>
<td>19.2</td>
<td>16.93</td>
<td>18.96</td>
<td>192.4</td>
</tr>
<tr>
<td>horizontal_view</td>
<td>3.77</td>
<td>0.88</td>
<td>12.59</td>
<td>56.24</td>
<td>16.64</td>
<td>2.02</td>
<td>32.64</td>
<td>19.81</td>
<td>21.84</td>
<td>253.8</td>
</tr>
<tr>
<td>rectangle</td>
<td>2.74</td>
<td>1.5</td>
<td>6.87</td>
<td>36.96</td>
<td>10.42</td>
<td>5.3</td>
<td>20.36</td>
<td>18.62</td>
<td>23.94</td>
<td>134.7</td>
</tr>
<tr>
<td>line</td>
<td>2.74</td>
<td>1.5</td>
<td>6.87</td>
<td>42.72</td>
<td>14.26</td>
<td>7.22</td>
<td>25.16</td>
<td>22.46</td>
<td>29.7</td>
<td>171.2</td>
</tr>
<tr>
<td>data</td>
<td>4.71</td>
<td>9.83</td>
<td>7.26</td>
<td>68.32</td>
<td>23.67</td>
<td>6.7</td>
<td>35.49</td>
<td>37.97</td>
<td>44.76</td>
<td>243.2</td>
</tr>
<tr>
<td>tree</td>
<td>2.86</td>
<td>1.97</td>
<td>5.29</td>
<td>65.5</td>
<td>10.49</td>
<td>3.26</td>
<td>50.3</td>
<td>38.31</td>
<td>41.59</td>
<td>373.1</td>
</tr>
<tr>
<td>string80_data</td>
<td>2.86</td>
<td>1.97</td>
<td>2.41</td>
<td>24.22</td>
<td>9.53</td>
<td>2.3</td>
<td>8.06</td>
<td>14.31</td>
<td>16.63</td>
<td>74.56</td>
</tr>
<tr>
<td>ratio</td>
<td>3.44</td>
<td>2.93</td>
<td>2.6</td>
<td>30.37</td>
<td>10.3</td>
<td>3.45</td>
<td>10.17</td>
<td>17.39</td>
<td>20.86</td>
<td>97.41</td>
</tr>
<tr>
<td>offset_ratio</td>
<td>3.57</td>
<td>0.59</td>
<td>1.48</td>
<td>17.59</td>
<td>5.9</td>
<td>1.65</td>
<td>2.99</td>
<td>9.24</td>
<td>10.89</td>
<td>52.12</td>
</tr>
<tr>
<td>list_data</td>
<td>2.86</td>
<td>1.97</td>
<td>4.33</td>
<td>57.82</td>
<td>29.69</td>
<td>3.26</td>
<td>35.9</td>
<td>37.35</td>
<td>40.63</td>
<td>320.3</td>
</tr>
<tr>
<td>integer_data</td>
<td>2.86</td>
<td>1.97</td>
<td>2.41</td>
<td>26.18</td>
<td>8.57</td>
<td>2.3</td>
<td>8.06</td>
<td>13.35</td>
<td>15.67</td>
<td>73.6</td>
</tr>
<tr>
<td>float_data</td>
<td>2.86</td>
<td>1.97</td>
<td>2.41</td>
<td>26.18</td>
<td>8.57</td>
<td>2.3</td>
<td>8.06</td>
<td>13.35</td>
<td>15.67</td>
<td>73.6</td>
</tr>
<tr>
<td>dictionary</td>
<td>2.86</td>
<td>1.97</td>
<td>2.41</td>
<td>23.26</td>
<td>11.45</td>
<td>2.3</td>
<td>15.74</td>
<td>14.31</td>
<td>16.63</td>
<td>105.3</td>
</tr>
<tr>
<td>boolean_data</td>
<td>2.86</td>
<td>1.97</td>
<td>2.41</td>
<td>26.18</td>
<td>8.57</td>
<td>2.3</td>
<td>8.06</td>
<td>13.35</td>
<td>15.67</td>
<td>73.6</td>
</tr>
<tr>
<td>context</td>
<td>1.52</td>
<td>1.95</td>
<td>4.72</td>
<td>48.27</td>
<td>20.44</td>
<td>11.27</td>
<td>17.98</td>
<td>31.22</td>
<td>42.57</td>
<td>128.8</td>
</tr>
<tr>
<td>action</td>
<td>2.68</td>
<td>4.83</td>
<td>8.18</td>
<td>42.7</td>
<td>14.3</td>
<td>4.55</td>
<td>21.44</td>
<td>21.23</td>
<td>26.82</td>
<td>109.4</td>
</tr>
<tr>
<td>quit</td>
<td>2.46</td>
<td>0.97</td>
<td>2.6</td>
<td>10.46</td>
<td>3.82</td>
<td>0.91</td>
<td>5.25</td>
<td>5.21</td>
<td>6.32</td>
<td>27.64</td>
</tr>
<tr>
<td>done</td>
<td>2.46</td>
<td>0.97</td>
<td>2.6</td>
<td>10.46</td>
<td>3.82</td>
<td>0.91</td>
<td>5.25</td>
<td>5.21</td>
<td>6.32</td>
<td>25.72</td>
</tr>
<tr>
<td>cancel</td>
<td>2.46</td>
<td>0.97</td>
<td>2.6</td>
<td>10.46</td>
<td>3.82</td>
<td>0.91</td>
<td>5.25</td>
<td>5.21</td>
<td>6.32</td>
<td>25.72</td>
</tr>
<tr>
<td>lims</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>32</td>
<td>7</td>
<td>2</td>
<td>10</td>
<td>14</td>
<td>18</td>
<td>84</td>
</tr>
</tbody>
</table>
Analysis

The above procedure was followed and produced the data shown below in Figure 5.1.

![R Squared Values for HCM]

Figure 5.1 R² Values for HCM

How does this analysis compare with the adjusted r² of the uncorrected values for the same metrics? For each base metric a regression was performed using the base metric as the independent variable and the change for that object as the dependent variable. The adjusted r² value for the regression for the original metric is shown below in Figure 5.2 compared to the r² values calculated for the HCM metrics as calculated above.
Figure 5.2 Adjusted $R^2$: HCM vs. Base Metric Only

**Resolve Question 1**

As can be seen from Figure 5.2 above, for every base metric HCM improved the correlation between the metric and maintenance effort. This is because a class's maintenance needs are influenced by its environment. Complex classes that have a small path length to the class do influence its maintenance needs. The OOP base metrics do not address this influence. Since HCM includes influences from both the base metric for a class and the influence of other classes, HCM should have a better correlation than base metrics alone. Based on the data shown in Figure 5.2 the conclusion is that HCM correlates with maintenance effort better than standard OOP metrics.
Review Question 2

Is it possible to improve the basic HCM formula and produce a better correlation to maintenance effort by weighting the ancestors or descendants of an object more than in standard HCM? It seems possible that an object's direct descendants or its ancestors would influence it more than for instance its siblings.

Procedure

Recall the definition of HCM from the previous chapter. However, there is a slight variation in how to calculate HCM, a new factor, \( w \), is added to the formula. This factor is used to weight an ancestor's contribution to the HCM for an object more than a standard object. This new metric will be called WHCM for Weighted HCM. Calculate WHCM for a given base metric \( b \) as follows:

\[
WHCM_j = \sum_{i=1}^{n} b_i \cdot s^p \cdot w_i
\]

where

- \( WHCM_j \) is the Hierarchy Corrected Metric for that class
- \( i \) is the number of classes in the hierarchy
- \( b \) is the base metric value for class \( i \)
- \( s \) is a scaling factor, and is always between 0 and 1.
- \( p \) is the path length between class \( i \) and the given class.
- \( w \) is the weight if class \( i \) is an ancestor of the given class, 1 otherwise.

Weighting a descendent is done similarly.
For example, calculate the WHCM for the three objects A, B, and Z show in Figure 5.3 above. The value for each object represents the calculated value for some base metric. The weighting used will be ancestor of weight 5. Based on the analysis done using varying scaling factors with the HCM calculation tool, it was shown that a scaling factor of 0.2 provides the best correlation for the UMIS data.

The WHCM value for class A, is the sum of three terms, the contribution from class A, from class B, and from class Z. From the WHCM definition above, we can calculate A's contribution as \((5 \cdot 0.2^0 \cdot 1)\) where 5 is the value of the base metric at A, 0.2 is the scaling factor, and 0 is the path length from A to A, and A is not an ancestor of itself, so the weighting factor is 1. The value of A's contribution is 5. The contribution from Z is \((15 \cdot 0.2^1 \cdot 5)\) where 15 is the base metric value for Z, 0.2 is the scaling factor, 1 is the path length from A to Z, and 5 is the weighting factor since Z is an ancestor of A. The value of Z's contribution is 15. The contribution from B is \((75 \cdot 0.2^2 \cdot 1)\) where 75 is the base metric value for B, 0.2 is the scaling factor, 2 is the path length from A to B, and 1 is
the weighting factor since B is not an ancestor of A. The value of B’s contribution is 3.

The final result:

\[ WHCM_A = (15 \cdot 0.2^1 \cdot 5) + (5 \cdot 0.2^0 \cdot 1) + (75 \cdot 0.2^2 \cdot 1) \]
\[ WHCM_A = 23 \]

In a similar manner WHCMs can be calculated for class B and class Z:

\[ WHCM_B = (15 \cdot 0.2^1 \cdot 5) + (5 \cdot 0.2^2 \cdot 1) + (75 \cdot 0.2^0 \cdot 1) \]
\[ WHCM_B = 90.2 \]
\[ WHCM_Z = (15 \cdot 0.2^0 \cdot 1) + (5 \cdot 0.2^1 \cdot 1) + (75 \cdot 0.2^1 \cdot 1) \]
\[ WHCM_Z = 31 \]

To resolve question 2 the following procedure is used. For each base metric, calculate four WHCM values, ancestor weighted 2 and 5, and descendent weighted 2 and 5. For each WHCM metric, perform a regression using the WHCM metric as the independent variable and the number of lines changed for that object as the dependent variable. The \( r^2 \) value generated by the regression measures correlation between the independent and dependent variable.

**Analysis**

The procedure described above was followed for ancestor weights of 2 and 5, and descendent weights of 2 and 5 also. The results are shown below in Figures 5.4 and 5.5.
Figure 5.4 Weighted Ancestor vs. Standard HCM

Figure 5.4 above shows the results of the above procedure for weighted ancestor. The standard HCM is shown in comparison to WHCM calculated with ancestor weights of 2 and 5.
Figure 5.5 Weighted Descendant vs. Standard HCM

Figure 5.5 above shows the results of the above procedure for weighted descendant. The standard HCM is shown in comparison to WHCM calculated with descendant weights of 2 and 5.

Resolve Question 2

From Figures 5.4 and 5.5 above it can be seen that for every base metric the adjusted $r^2$ is greater (or very close to in the case of DIT on weighted ancestor) for the standard HCM than for any of the WHCMs. This suggests that special relationships between objects such as ancestor and descendant are not important in calculating HCM. Consider the class hierarchy shown in Figure 5.6 below. An ancestor WHCM metric value for class D would give a greater weight to the influence of class A than to the influence of class E.
What the results shown in Figures 5.4 and 5.5 above demonstrate is that giving extra weight in the calculation of the HCM metric to the classes that have a special relationship with the class in question does not increase the correlation to maintenance needs, in fact, it seems to decrease the correlation. The only important factors are the class's base metric value and the path length. So if class E and class A both had the same value for the base metric being used, they both would influence D by the same amount since their path length to D of 2 is the same. The fact that A is an ancestor of D is not important in this calculation.
Summary

In this chapter HCM was validated by showing it correlates with maintenance effort better than standard OOP metrics. It was also shown that weighting the input of ancestor and descendent objects does not improve HCM's correlation with maintenance effort.
Chapter 6  Conclusions and Future Research

Introduction

This research explored software product metrics in the Object Oriented Paradigm. Specifically, it addressed the lack of emphasis in existing OOP metrics on a class's location in the inheritance hierarchy.

Summary of Research

A new OOP metric was proposed, Hierarchy Corrected Metric (HCM). HCM attempts to quantify the influence that objects near to a given object in the inheritance hierarchy have on the maintenance needs of that object. It was also hypothesized that an object's direct descendants or ancestors would have a greater influence on an object's maintenance needs than other objects in the hierarchy.

Conclusions

Using the maintenance data from the UMIS system, HCM was shown to have a higher correlation to maintenance effort than any base metric. HCM is therefore a good predictor of maintenance effort. Also using the maintenance data from the UMIS system,
it was shown that a class's direct descendants or ancestors do not have any more of an influence on an class's maintenance needs than other classes in the inheritance hierarchy.
Future Research

There was another SPS developed system that Li collected data on, QUES. QUES was developed after UMIS had been deployed, and reused many of the classes from UMIS. More than two thirds of the classes in QUES were classes that were reused from UMIS. Preliminary analysis on the QUES data revealed that HCM performed poorly on the data from QUES. The reason that is since the QUES data included many classes that showed no changes between QUES releases (since they were originally UMIS classes that had already had needed maintenance performed on them) that HCM would not function properly since it is a metric that is based on the entire hierarchy. Kafura [KAFD95] suggested a modification to the HCM formula that would include a factor for each class that would reflect that classes maturity level (a measure of how much maintenance has already been performed on that class).

HCM has only been shown to correlate well with maintenance effort for a single system. Further verification of the HCM metric is needed by analysis of its use on additional systems.

The definition of HCM should also be expandable to address languages that allow multiple inheritance.

As new base metrics are introduced for the OOP paradigm, HCM should be used with the new metric as its base to see if an even greater correlation can be achieved.
Bibliography


[KAFD95] Kafura, Dennis, Personal Communication, October 1995


Vita

Cary Douglas Long was born on Feb. 18, 1966 in Kansas City, Mo. He completed his undergraduate double major in Computer Science and Philosophy from Virginia Polytechnic Institute and State University in 1988. He started his search for the elusive Master's degree also in 1988, and also at Virginia Polytechnic Institute and State University. It only took him 7 years to finally finish the thing after getting married, having a child, starting his first real job, getting divorced, changing major professors long distance, taking off for a year or so in the middle of working on the thesis... twice. What a long strange journey it has been. That Ph.D. shouldn't take more than another 5 years, ten years tops. He finds writing about himself in the third person very strange.