Title: Applying and Interpreting Object Oriented Metrics
Presenter: Dr. Linda H. Rosenberg
Track: Track 7 - Measures/Metrics
Day: Wednesday
Keywords: Metrics, Object-Oriented

Abstract: Object-oriented design and development is becoming very popular in today's software development environment. Object oriented development requires not only a different approach to design and implementation, it requires a different approach to software metrics. Since object oriented technology uses objects and not algorithms as its fundamental building blocks, the approach to software metrics for object oriented programs must be different from the standard metrics set. Some metrics, such as lines of code and cyclomatic complexity, have become accepted as "standard" for traditional functional/ procedural programs, but for object-oriented, there are many proposed object oriented metrics in the literature. The question is, "Which object oriented metrics should a project use, and can any of the traditional metrics be adapted to the object oriented environment?"

In this paper, the Software Assurance Technology Center (SATC) at NASA Goddard Space Flight Center discusses its approach to choosing metrics for a project by first identifying the attributes associated with object oriented development. Within this framework, nine metrics for object oriented are selected. These metrics include three traditional metrics adapted for an object oriented environment, and six "new" metrics to evaluate the principle object oriented structures and concepts. The metrics are first defined, then using a very simplistic object oriented example, the metrics are applied. Interpretation guidelines are then discussed and data from NASA projects are used to demonstrate the application of the metrics.

In the experience of the SATC, projects choose the data they collect by default - if the tool they are using compiles it, the project collects it. The purpose of this paper is to help project managers choose a comprehensive set of metrics, not by default, but by using a set of metrics based on attributes and features of object oriented technology.

Applying and Interpreting Object Oriented Metrics

1. INTRODUCTION

Object-oriented design and development are popular concepts in today's software development environment. They are often heralded as the silver bullet for solving software problems, while in reality there is no silver bullet; object oriented development has proved its value for systems that must be maintained and modified. Object oriented software development requires a different approach from more traditional functional decomposition and data flow development methods. While the functional and data flow approaches commence by considering the systems behavior and/or data separately, object oriented analysis approaches the problem by looking for system entities that combine them. Object oriented analysis and design focuses on

objects as the primary agents involved in a computation; each class of data and related operations are collected into a single system entity.

This paper will first briefly discuss nine metrics currently being applied by the SATC to NASA object oriented projects. These include three "traditional" metrics adapted for an object oriented environment, and six "new" metrics to evaluate the principle object oriented structures and concepts. The SATC's approach to identifying a set of object oriented metrics was to identify the primary critical constructs of object oriented design and to select metrics that evaluate those areas. The metrics focus on internal object structures that reflect the complexity of each individual entity, such as methods and classes, and on external complexity that measures the interactions among entities, such as coupling and inheritance. The metrics measure computational complexity that affects the efficiency of an algorithm and the use of machine resources, as well as psychological complexity factors that affect the ability of a programmer to create, comprehend, modify and maintain software.

But as important as the metrics chosen is what the metrics "tell" the developers and managers about the quality and object oriented structure of the design and code; metrics without interpretation guidelines are of little value. Metrics for object oriented development is a relatively new field of study, however, and have not reached maturity. Although some numeric thresholds are suggested by analysis developers, there is little application data to justify specific "good" and "bad" ranges. Knowledge and experience of the programmers, managers, researchers and SATC staff currently serve as the basis for the interpretation guidelines of the metric analysis presented in this paper. As each metric is defined, guidelines for interpreting the values are suggested. In many cases, however, to improve one metric means a trade-off with another.

This paper starts with an overview of the metrics recommended by the SATC for object oriented systems. These metrics include modifications of “traditional” metrics as well as “new” metrics for specific object oriented structures. Since the object oriented metrics require a cursory understanding of the object oriented concepts, Section 3 presents a pictorial representation of the basic object oriented structures and defines the key terms. In Section 5, we discuss the metrics in-depth. The design for a simple object oriented example is used to demonstrate the metric calculations. In Section 5, interpretation guidelines are discussed. Then in Section 6 the applications and interpretations of the metrics are demonstrated using NASA project data.

2. OVERVIEW - OBJECT ORIENTED METRICS

In this paper, the SATC discusses its applied research of object oriented metrics. The research was done by surveying the literature on object oriented metrics and then applying the SATC experience in traditional software metrics to select the object oriented metrics that support the goal of measuring design and code quality. In addition, we required that a metric be feasible to compute and have a clear relationship to the object oriented structures being measured. At this time, many object oriented metrics proposed in the literature lack a theoretical basis, while others have not yet been validated. Some of these metrics are very labor intensive to collect, or are dependent on the implementation environment. The object oriented metrics applied by the SATC are computable, can be related to desirable software qualities, and are in the process of being validated.

The SATC's approach to identifying a set of object oriented metrics was to focus on the primary, critical constructs of object oriented design and to select metrics that apply to those areas. The suggested metrics are supported by most literature and some object oriented tools. The metrics evaluate the object oriented concepts: methods, classes, coupling, and inheritance. The metrics focus on internal object structure that reflects the complexity of each individual entity and on external complexity that measures the interactions among entities. The metrics measure computational complexity that affects the efficiency of an algorithm and the use of machine resources, as well as psychological complexity factors that affect the ability of a programmer to create, comprehend, modify, and maintain software.

We support the use of three traditional metrics and present six additional metrics specifically for object oriented systems. The SATC has found that there is considerable disagreement in the field about software quality metrics for object oriented systems [2, 6]. Some researchers and practitioners contend traditional metrics are inappropriate for object oriented systems. There are valid reasons for applying traditional metrics, however, if it can be done. The traditional metrics have been widely used, they are well understood by researchers and practitioners, and their relationships to software quality attributes have been validated [2, 6, 11, 12].

Table 1 presents an overview of the metrics applied by the SATC for object oriented systems. The SATC supports the continued use of traditional metrics, but within the structures and confines of object oriented systems. The first three metrics in Table 1 are examples of traditional metrics applied to the object oriented structure of methods instead of functions or procedures. The next six metrics are specifically for object oriented systems and the object oriented construct applicable is indicated.

SOURCE	METRIC	OBJECT-ORIENTED CONSTRUCT
Traditional	Cyclomatic complexity (CC)	Method
Traditional	Lines of Code (LOC)	Method
Traditional	Comment percentage (CP)	Method
NEW Object-Oriented	Weighted methods per class (WMC)	Class/Method
NEW Object-Oriented	Response for a class (RFC)	Class/Message
NEW Object-Oriented	Lack of cohesion of methods (LCOM)	Class/Cohesion
NEW Object-Oriented	Coupling between objects (CBO)	Coupling
NEW Object-Oriented	Depth of inheritance tree (DIT)	Inheritance
NEW Object-Oriented	Number of children (NOC)	Inheritance

Table 1: SATC Metrics for Object Oriented Systems

3. OVERVIEW - OBJECT ORIENTED STRUCTURES

A brief description of object oriented structures is given in this section using the pictorial description in Figure 1 and the definitions in Table 2.

Attribute	Defines the structural properties of classes, unique within a class, generally a noun.
Class	A set of objects that share a common structure and common behavior manifested by a set of methods; the set serves as a template from which object can be instantiated (created).
Cohesion	The degree to which the methods within a class are related to one another.
Coupling	Object X is coupled to object Y if and only if X sends a message to Y.
Inheritance	A relationship among classes, wherein an object in a class acquires characteristics from one or more other classes.
Instantiation	The process of creating an instance of the object and binding or adding the specific data.
Message	A request that an object makes of another object to perform an operation.
Method	An operation upon on object, defined as part of the declaration of a class.
Object	An instantiation of some class which is able to save a state (information) and which offers a number of operations to examine or affect this state.
Operation	An action performed by or on an object, available to all instances of class, need not be unique.

Table 2: Key Object Oriented Terms for Metrics

The new object oriented development methods have their own terminology to reflect the new structural concepts. Referencing Figure 1, an object oriented system starts by defining a *class* (Class A) that contains related or similar *attributes* and *operations* (some operations are *methods*). The classes are used as the basis for *objects* (Object A1). A child class *inherits* all of the attributes and operations from its parent class, in addition to having its own attributes and operations. A child class can also become a parent class for other classes, forming another branch in the hierarchical tree structure. When an object is created to contain data or information, it is an *instantiation* of the class. Classes interact or communicate by passing *messages*. When a message is passed between two classes, the classes are *coupled*. These specific terms are defined in Table 2 [1, 4, 9].

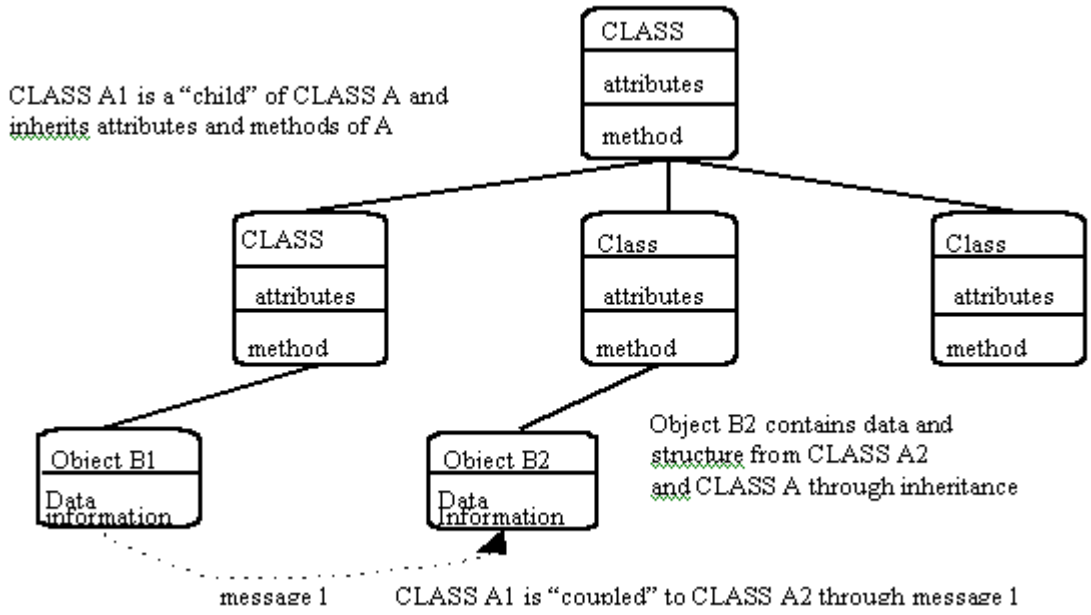


Figure 1: Pictorial Description of Object Oriented Terms

Figure 2 is an example application with 3 classes; the root or main class, *Store_de*pt and two child classes, *Clothing* and *Appliances*. Each department will have *a Manager, # Employees* and *Floor space*; each child class inherits the attributes of *Manager, # Employees, Floor Space* from *Store_dept*. The class *Clothing* will have additional attributes of *Customer Gender, Size range* and *Specialty*. The class of *Appliance* Departments will have an attribute of *Category*. Specific named departments are objects. Objects of the *Class Clothing* are the *Toddlers Department* and *Men's Suits Department*. In the *Appliances* class, the objects are *Large Appliance Department, Small Kitchen Appliances*, and the *Electronics Department*.

Methods are operations done on an object. Examples of what all store departments need to do are *Display merchandise*, *Give credit*, and *Exchange merchandise*. The *Clothing Department* will inherit these methods but also have *Dressing rooms*. The *Appliance Department* will also have *Delivery and Install, Service, Parts Ordering* and *Technical support.*

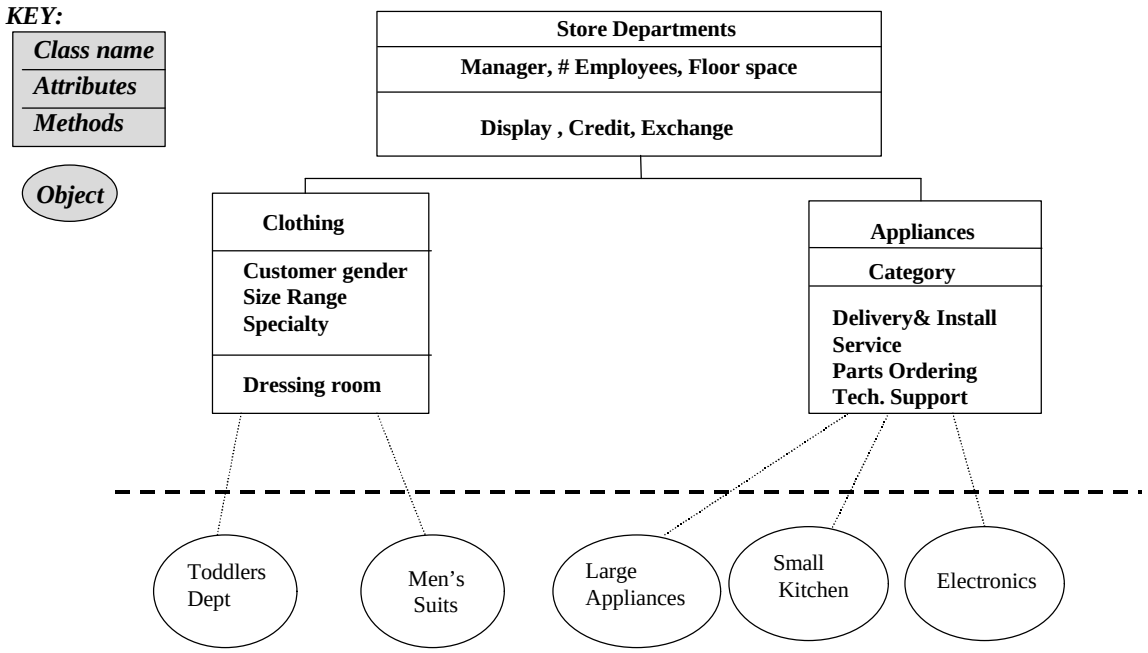


Figure 2: Example Application

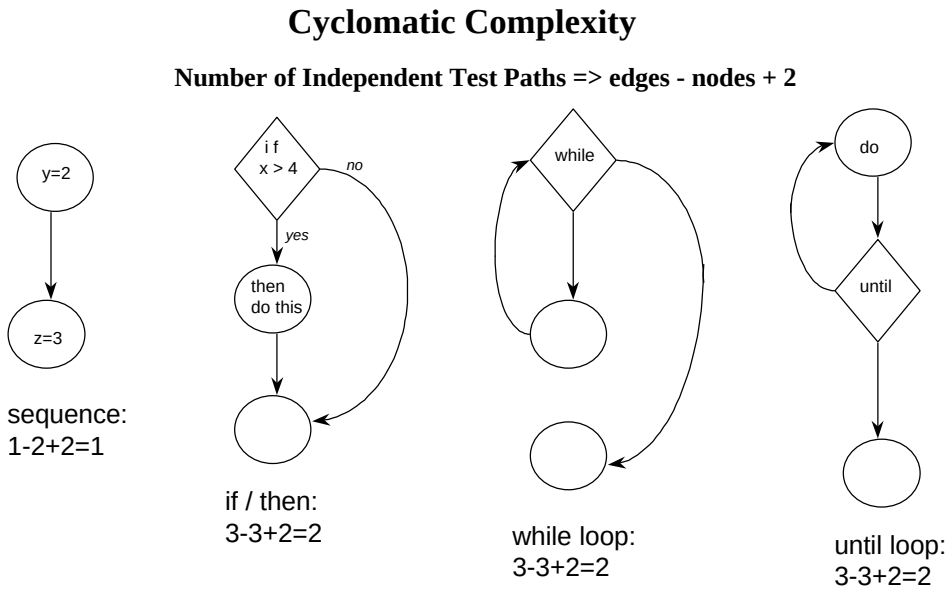
4. METRICS FOR OBJECT ORIENTED SYSTEMS

A. Traditional Metrics

There are many metrics that are applied to traditional functional development. The SATC, from experience, has identified three of these metrics that are applicable to object oriented development: Complexity, Size, and Readability. To measure the complexity, the cyclomatic complexity is used.

A.1 METRIC 1: Cyclomatic Complexity (CC)

Cyclomatic complexity (McCabe) is used to evaluate the complexity of an algorithm in a method. It is a count of the number of test cases that are needed to test the method comprehensively. The formula for calculating the cyclomatic complexity is the number of edges minus the number of nodes plus 2. For a sequence where there is only one path, no choices or option, only one test case is needed. An *IF* loop however, has two choices, if the condition is true, one path is tested; if the condition is false, an alternative path is tested. Figure 3 shows



examples of calculations for the cyclomatic complexity for four basic programming structures. [7]

Figure 3 :Example Calculations Cyclomatic Complexity

A method with a low cyclomatic complexity is generally better. This may imply decreased testing and increased understandability or that decisions are deferred through message passing, not that the method is not complex. Cyclomatic complexity cannot be used to measure the complexity of a class because of inheritance, but the cyclomatic complexity of individual methods can be combined with other measures to evaluate the complexity of the class. Although this metric is specifically applicable to the evaluation of Complexity, it also is related to all of the other attributes [3, 5, 6, 7, 11].

A.2 METRIC 2: Size

Size of a class is used to evaluate the ease of understanding of code by developers and maintainers. Size can be measured in a variety of ways. These include counting all physical lines of code, the number of statements, the number of blank lines, and the number of comment lines. Lines of Code(LOC) counts all lines. Non-comment Non-blank (NCNB) is sometimes referred to as Source Lines of Code and counts all lines that are not comments and not blanks. Executable Statements (EXEC) is a count of executable statements regardless of number of physical lines of code. For example, in FORTRAN and *IF* statement may be written:

IF X = 3

Then

Y = 10

This example would be 3 LOC, 3 NCNB, and 1 EXEC.

Executable statements is the measure least influenced by programmer or language style. Therefore, since NASA programs are frequently written using multiple languages, the SATC uses executable statements to evaluate project size. [10]

Thresholds for evaluating the meaning of size measures vary depending on the coding language used and the complexity of the method. However, since size affects ease of understanding by the developers and maintainers, classes and methods of large size will always pose a higher risk. [3, 6; 7, 11]

A.3 METRIC 3: Comment Percentage

The line counts done to compute the Size metric can be expanded to include a count of the number of comments, both on-line (with code) and stand-alone. The comment percentage is calculated by the total number of comments divided by the total lines of code less the number of blank lines. Since comments assist developers and maintainers, higher comment percentages increase understandability and maintainability. [10]

B. Object-Oriented Specific Metrics

As discussed, many different metrics have been proposed for object oriented systems. The object oriented metrics that were chosen by the SATC measure principle structures that, if improperly designed, negatively affect the design and code quality attributes.

The selected object oriented metrics are primarily applied to the concepts of classes, coupling, and inheritance. Preceding each metric, a brief description of the object oriented structure is given. For some of the object-oriented metrics discussed here, multiple definitions are given; researchers and practitioners have not reached a common definition or counting methodology. In some cases, the counting method for a metric is determined by the software analysis package being used to collect the metrics.

Recall, a class is a template from which objects can be created. This set of objects shares a common structure and a common behavior manifested by the set of methods. A method is an operation upon an object and is defined in the class declaration. A message is a request that an object makes of another object to perform an operation. The operation executed as a result of receiving a message is called a method. Cohesion is the degree to which methods within a class are related to one another and work together to provide well-bounded behavior. Effective object oriented designs maximize cohesion because cohesion promotes encapsulation. Coupling is a measure of the strength of association established by a connection from one entity to another. Classes (objects) are coupled when a message is passed between objects; when methods declared in one class use methods or attributes of another class. Inheritance is the hierarchical relationship among classes that enables programmers to reuse previously defined objects including variables and operators. [2, 3, 5, 8]

Figure 2 is duplicated here as Figure 4 to use as an example application to demonstrate how these metrics would be calculated.

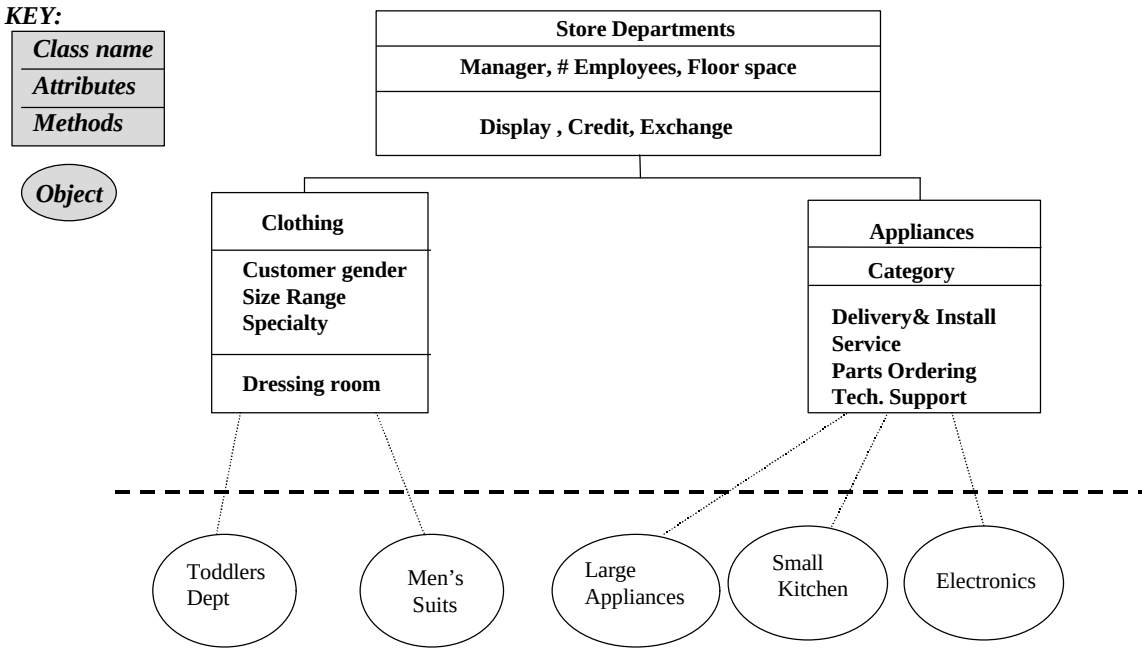


Figure 4: Object Oriented Application Example

B.1 METRIC 4: Weighted Methods per Class (WMC)

The WMC is a count of the methods implemented within a class or the sum of the complexities of the methods (method complexity is measured by cyclomatic complexity). The second measurement is difficult to implement since not all methods are assessable within the class hierarchy due to inheritance. The number of methods and the complexity of the methods involved is a predictor of how much time and effort is required to develop and maintain the class. The larger the number of methods in a class, the greater the potential impact on children; children inherit all of the methods defined in the parent class. Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse. [2, 6, 7, 8]

Referring to Figure 4, WMC is calculated by counting the number of methods in each class, therefore:

WMC for *Clothing_dept* = 1
WMC for *Appliance_dept* = 4

B.2 METRIC 5: Response for a Class (RFC)

The RFC is the count of the set of all methods that can be invoked in response to a message to an object of the class or by some method in the class. This includes all methods accessible within the class hierarchy. This metric looks at the combination of the complexity of a class through the number of methods and the amount of communication with other classes. The larger the number of methods that can be invoked from a class through messages, the greater the complexity of the class. If a large number of methods can be invoked in response to a

message, the testing and debugging of the class becomes complicated since it requires a greater level of understanding on the part of the tester. A worst case value for possible responses will assist in the appropriate allocation of testing time. [2, 6, 7, 8]

The RFC for *Store_dept* in Figure 4 is the number of methods that can be invoked in response to messages by itself (*Store_dept*), by *Clothing_dept*, and by *Appliance_dept*.

The RFC for *Store_dept* = 3 (self) + 1 (*Clothing_dept*) + 4 (*Appliance_dept*) = 8

B.3 Metric 6 – Lack of Cohesion (LCOM)

Lack of Cohesion (LCOM) measures the dissimilarity of methods in a class by instance variable or attributes. A highly cohesive module should stand alone; high cohesion indicates good class subdivision. Lack of cohesion or low cohesion increases complexity, thereby increasing the likelihood of errors during the development process. High cohesion implies simplicity and high reusability. High cohesion indicates good class subdivision. Lack of cohesion or low cohesion increases complexity, thereby increasing the likelihood of errors during the development process. Classes with low cohesion could probably be subdivided into two or more subclasses with increased cohesion. [2, 3, 6; 7, 8] Figure 5 is an alternative program design for the segment in Figure 4 and useful in demonstrating cohesion.

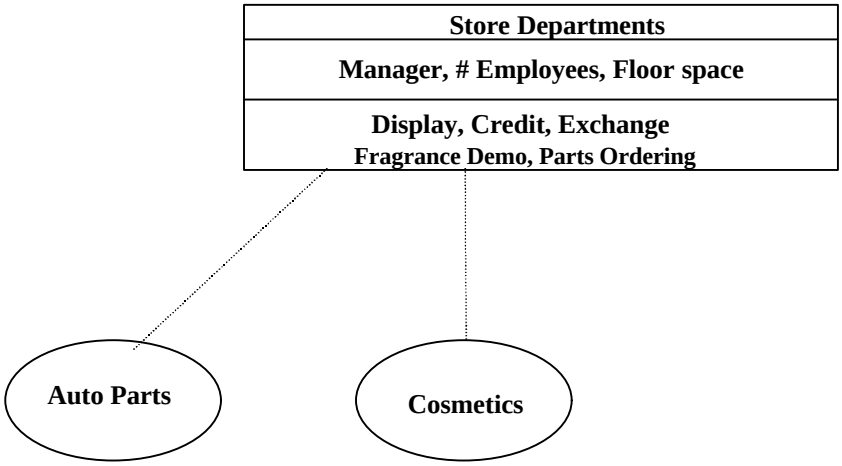


Figure 5: Alternative Design

In Figure 5, the two child classes of *Clothing* and *Appliances* have been eliminated, the attributes and methods combined into one class *Store_dept*. Figure 5 shows a design with high lack of cohesion because there are relatively few common attributes and methods among the objects. *Auto_Parts* needs the method *Parts_Ordering* but not *Fragrence_Demonstrations*. *Cosmetics* needs *Fragrence_Demonstrations* but not Parts_Ordering. Because the objects have few methods in common, there is a high lack of cohesion. This implies further abstraction is needed – similar objects need to be grouped together by creating child classes for them.

B.4 METRIC 7: Coupling Between Object Classes (CBO)

Coupling Between Object Classes (CBO) is a count of the number of other classes to which a class is coupled. It is measured by counting the number of distinct non-inheritance related class hierarchies on which a class depends. Excessive coupling is detrimental to modular

design and prevents reuse. The more independent a class is, the easier it is reuse in another application. The larger the number of couples, the higher the sensitivity to changes in other parts of the design and therefore maintenance is more difficult. Strong coupling complicates a system since a class is harder to understand, change or correct by itself if it is interrelated with other classes. Complexity can be reduced by designing systems with the weakest possible coupling between classes. This improves modularity and promotes encapsulation. [2, 3, 5, 6, 7, 8]

Figure 6 is an exaggerated example of high coupling between objects. Here two departments are identified as *Jackets* and *Trousers*. They both have the same attributes and the same methods. This implies that the design in Figure 6 is probably not the most efficient design and these departments should be combined into one class.

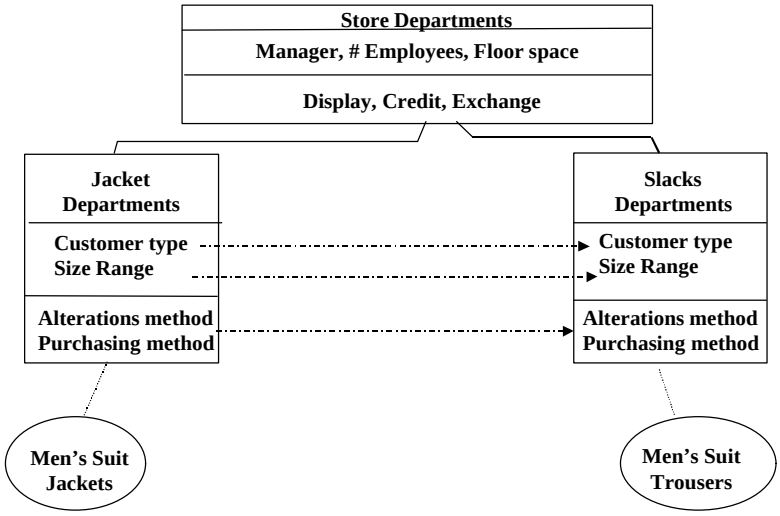


Figure 6: Example of Excessive Coupling

B.5 METRIC 8: Depth of Inheritance Tree (DIT)

The depth of a class within the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree and is measured by the number of ancestor classes. The deeper a class is within the hierarchy, the greater the number methods it is likely to inherit making it more complex to predict its behavior. Deeper trees constitute greater design complexity, since more methods and classes are involved, but the greater the potential for reuse of inherited methods. A support metric for DIT is the number of methods inherited (NMI). [2, 3, 6, 7, 8]

In Figure 4, *Store_Dept* is the root and has a DIT of 0. The DIT for *Clothing* is 1.

B.6 METRIC 9: Number of Children (NOC)

The number of children is the number of immediate subclasses subordinate to a class in the hierarchy. It is an indicator of the potential influence a class can have on the design and on the system. The greater the number of children, the greater the likelihood of improper abstraction of the parent and may be a case of misuse of subclassing. But the greater the number

of children, the greater the reuse since inheritance is a form of reuse. If a class has a large number of children, it may require more testing of the methods of that class, thus increase the testing time. [2, 6, 7, 8]

In Figure 4, *Store_Dept* has an NOC of 2. NOC for *Clothing* is 0 since it is a terminating or leaf node in the tree structure.

5. INTERPRETATION GUIDELINES

While it is interesting to propose a set of metrics for object oriented system, the value of the metrics is in their application to programs – how can they help developers improve the quality of the programs? While there are many guidelines as to how to interpret the metrics, there is insufficient statistical data to prove that a value of 8 for one metric is twice as complex or twice as "bad" as a value of 4. The SATC therefore, proposes interpretation guidelines based on a comparison of the values, looking at the outliers to determine why they are different from the other modules of code. This is not an indication of "badness" but an indicator of difference that needs to be investigated. Table 3 is a summary of the objectives for the values suggested above in the description of the metrics.

METRIC	**OBJECTIVE**
Cyclomatic Complexity	Low
Lines of Code/Executable Statements	Low
Comment Percentage	~ 20 – 30 %
Weighted Methods per Class	Low
Response for a Class	Low
Lack of Cohesion of Methods Cohesion of Methods	Low High
Coupling Between Objects	Low
Depth of Inheritance	Low (trade-off)
Number of Children	Low (trade-off)

Table 3 : Interpretation Guidelines

However, as indicated in the last two metrics, there is a trade-off with many of the metrics. A high Depth in Tree will increase maintainability complexity but also shows increased reuse. A high number of children will increase testing efforts but will also accompany increased the extent of reuse efficiency. A developer must be aware of the relationships of the structures and that altering the size of one metric can impact areas such as testing, understandability, maintainability, development effort and reuse. [10]

6. APPLICATION

For some of the metrics, a simple histogram demonstrates prevailing and extreme values. For the this project in Figure 7, a histogram of Weighted Methods per Class (WMC) reveals that, while most classes have a WMC of less than 20, there are a few classes with WMC greater than

100. Those few classes with the highest WMC are candidates for inspection and/or revision. This histogram is also useful for monitoring complexity over time.

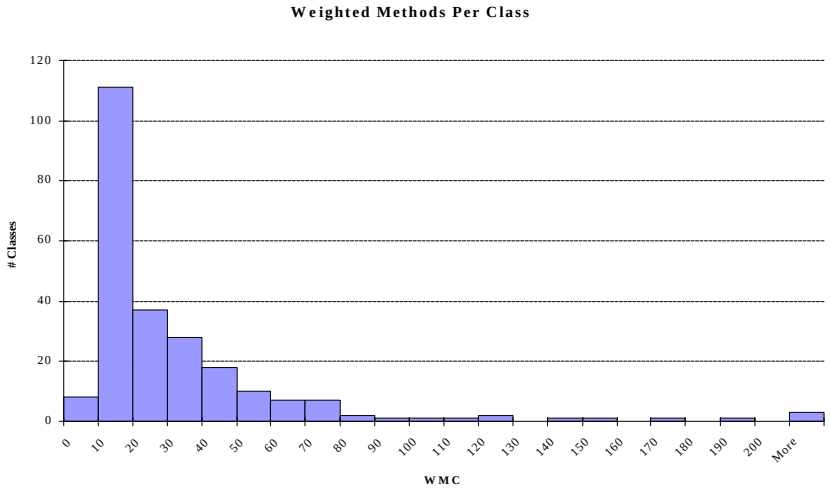


Figure 7: Weighted Methods Per Class

There are a few classes in the project shown in Figure 8 that are capable of invoking more than 200 methods. Classes with large RFC have a greater complexity and decreased understandability. Testing and debugging are more complicated. This information is useful when monitored over time also.

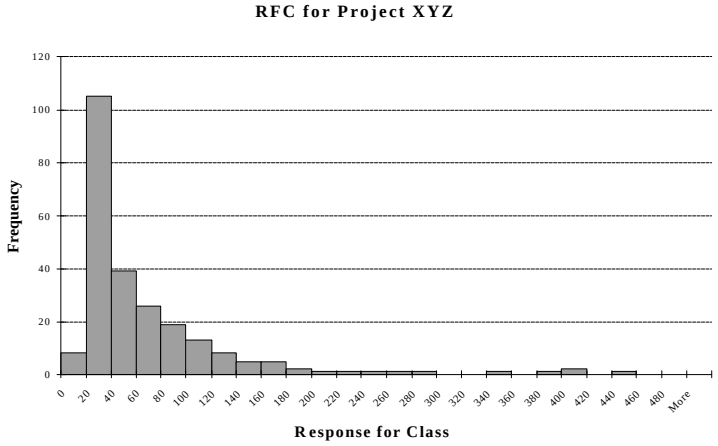


Figure 8: Response for a Class

Figure 9 examines the Coupling Between Objects (CBO). Of the 240 classes in this project, more than a third are self-contained. Higher CBO indicates classes that may be difficult to understand, less likely for reuse and more difficult to maintain.

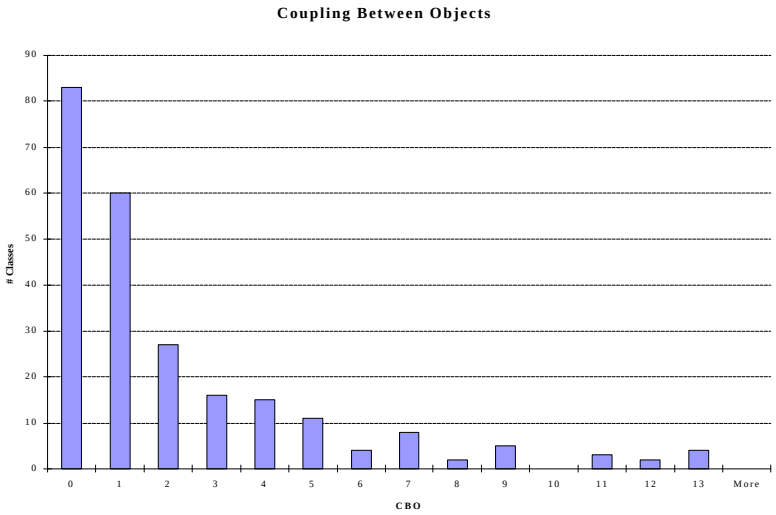


Figure 9: Coupling Between Objects

The metrics for the hierarchical structure, Number of Children (NOC) and Depth in Tree (DIT) can also be graphically depicted as shown in Figure 10. A class with DIT = 0 is the "root" of a hierarchy. If it is also a "leaf", NOC = 0, then it is standalone code that does not benefit from inheritance or reuse. Almost 66% of this project's classes are below other classes in the tree, which indicates a moderate level of reuse. Higher percentages for DIT's of 2 and 3 would show a higher degree of reuse, but increased complexity.

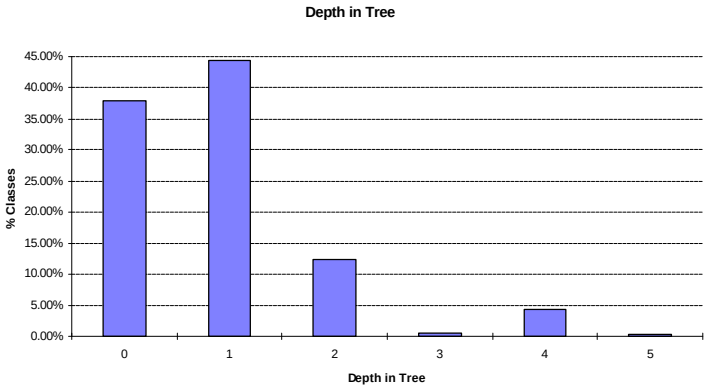


Figure 10: Depth in Tree (DIT)

The value of Lack of Cohesion of Methods (LCOM) depends on the number of methods, so there is a maximum value possible. Figure 11 is a plot of measured LCOM compared to possible maximums. While there is little experience with the LCOM metric, intuition says that the smaller actual LCOM is compared to its possible maximum, the better. In Figure 11 we look

at LCOM to identify the values closest to the line. The SATC also uses the trend line shown in the graph to make comparisons between projects and between languages.

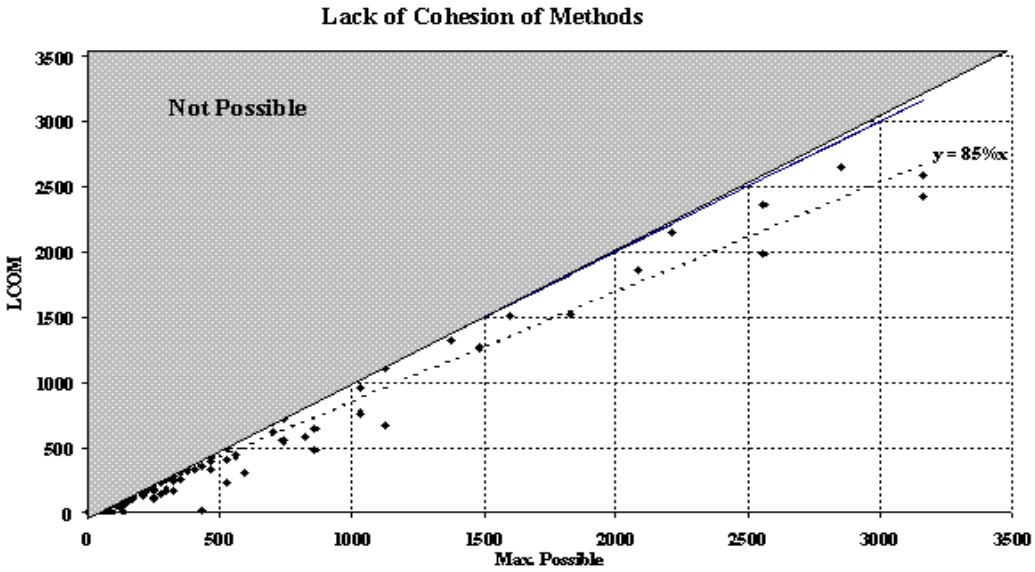
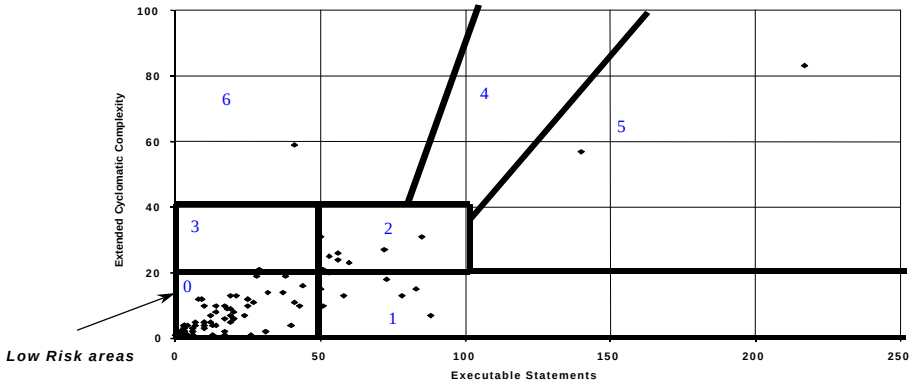


Figure 11: Lack of Cohesion of Methods

For many of the metrics, it is more effective to analyze the modules using two metrics. In Figure 12 the methods are plotted based on size and complexity. The SATC has done extensive applied research to identify the preferred values. The "risk regions" shown indicate where methods have the potential for poor quality that will effect maintainability, reusability and readability. (These regions of risk were developed for non object oriented code and are expected to decrease in size with further research.) The table below the graph summarizes the diagram.



	Risk 0	Risk 1	Risk 2	Risk 3	Risk 4	Risk 5	Risk 6	Total
Count	85	7	9	1	0	2	1	105
Percent	81.0%	6.7%	8.6%	1.0%	0.0%	1.9%	1.0%	100.0%

Higher risk due to size and complexity

Figure 12: Size to Complexity Comparison

In Figure 13, Response for a Class is plotted against the number of methods. Points on or near the "possible" line represent classes that do not invoke outside methods. This indicates to developers that there are some classes with more than 40 methods that also affect many objects in other classes. These are prime candidates for walk-throughs and testing.

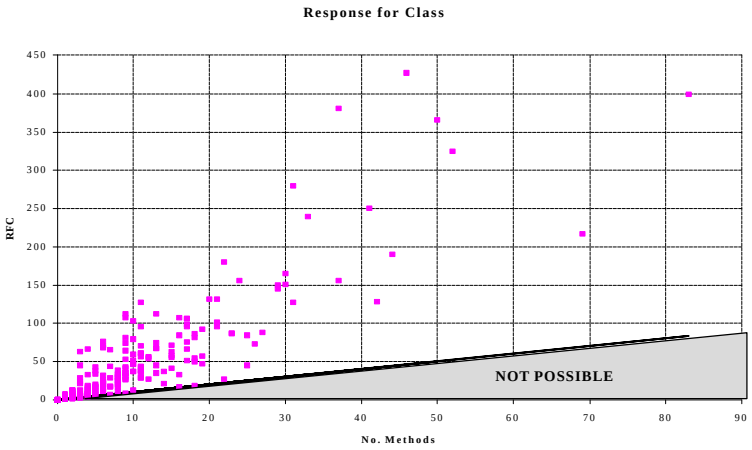


Figure 13: Number of Methods by Response for Class

As discussed, there is a trade-off when determining the appropriate number of children and the depth of the tree. Higher DIT's indicate a trade-off between increased complexity and increased reuse. Higher NOC's also indicate reuse, but may require more testing. Figure 14 demonstrates how the two-way view of the data identifies an interesting class – one that is three steps down form the root and has 40 children.

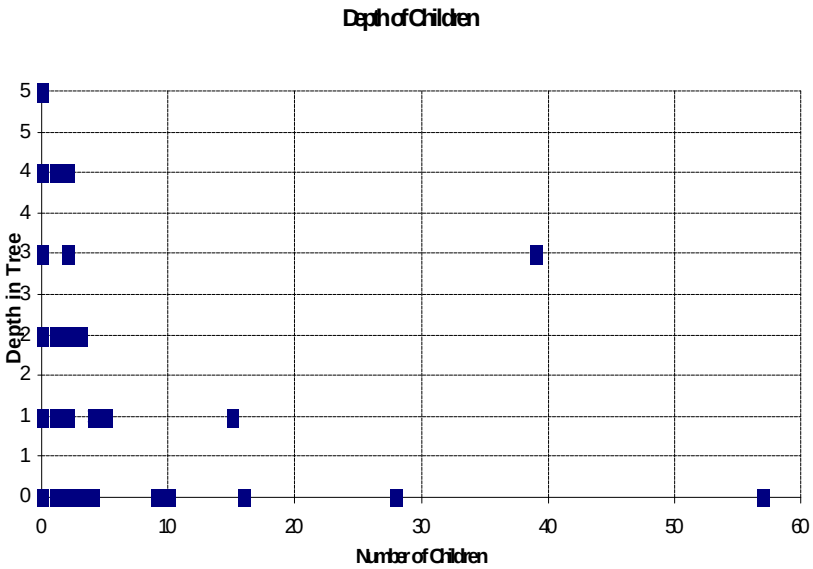


Figure 14: Hierarchical Evaluation

7. SUMMARY

Object oriented metrics help evaluate the development and testing efforts needed, the understandability, maintainability and reusability. This information is summarized in Table 4.

Metrics	Objective	Testing Efforts	Understan -dability	Maintain -ability	Develop Effort	Reuse
Complexity	↓	↓	↑	↑		
Size (LOC)	↓	↓	↑	↑		
Comment %	↑	↓	↑	↑	↓	
WMC	↓			↑	↓	↑
RFC	↓	↓				↑
LCOM	↓		↑	↑	↓	↑
CBO	↓	↓	↑	↑		↑

Table 4 : Object Oriented Metrics Effects

8. CONCLUSION

Object oriented metrics exist and do provide valuable information to object oriented developers and project managers. The SATC has found that a combination of "traditional" metrics and metrics that measure structures unique to object oriented development is most effective. This allows developers to continue to apply metrics that they are familiar with, such as complexity and lines of code to a new development environment. However, now that new concepts and structures are being applied, such inheritance, coupling, cohesion, methods and classes, metrics are needed to evaluate the effectiveness of their application. Metrics such as Weighted Methods per Class, Response for a Class, and Lack of Cohesion are applied to these areas. The application of a hierarchical structure also needs to be evaluated through metrics such as Depth in Tree and Number of Children.

At this time there are no clear interpretation guidelines for these metrics although there are guidelines based on common sense and experience.

9. REFERENCES

1. Booch, Grady, *Object Oriented Analysis and Design with Applications*, The Benjamin/Cummings Publishing Company, Inc., 1994.
2. Chidamber, Shyam and Kemerer, Chris, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, June, 1994, pp. 476-492.
3. Hudli, R., Hoskins, C., Hudli, A., "Software Metrics for Object Oriented Designs", IEEE, 1994.
4. Jacobson, Ivar, *Object Oriented Software Engineering, A Use Case Driven Approach*, Addison-Wesley Publishing Company, 1993.
5. Lee, Y., Liang, B., Wang, F., "Some Complexity Metrics for Object Oriented Programs Based on Information Flow", *Proceedings: CompEuro*, March, 1993, pp. 302-310.

6. Lorenz, Mark and Kidd, Jeff, *Object Oriented Software Metrics*, Prentice Hall Publishing, 1994.
7. McCabe & Associates, *McCabe Object Oriented Tool User's Instructions*, 1994.
8. Rosenberg, Linda H., "Metrics for Object Oriented Environments", EFAITP/AIE Third Annual Software Metrics Conference, December, 97.
9. Sommerville, Ian, *Software Engineering*, Addison-Wesley Publishing Company, 1992.
10. Sharble, Robert, and Cohen, Samuel, "The Object Oriented Brewery: A Comparison of Two object oriented Development Methods", *Software Engineering Notes*, Vol 18, No 2., April 1993, pp 60 -73.
11. Tegarden, D., Sheetz, S., Monarchi, D., "Effectiveness of Traditional Software Metrics for Object Oriented Systems", *Proceedings: 25th Hawaii International Conference on System Sciences*, January, 1992, pp. 359-368.
12. Williams, John D., "Metrics for Object Oriented Projects", *Proceedings: ObjectExpoEuro Conference*, July, 1993, pp. 13-18.

10. BIOGRAPHIES

Linda H. Rosenberg, Ph.D.

Dr. Rosenberg is an Engineering Section Head at Unisys Government Systems in Lanham, MD. She is contracted to manage the Software Assurance Technology Center (SATC) through the System Reliability and Safety Office in the Flight Assurance Division at Goddard Space Flight Center, NASA, in Greenbelt, MD. The SATC has four primary responsibilities: Metrics, Standards and Guidance, Assurance tools and techniques, and Outreach programs. Although she oversees all work areas of the SATC, Dr. Rosenberg's area of expertise is metrics. She is responsible for overseeing metric programs to establish a basis for numerical guidelines and standards for software developed at NASA, and to work with project managers to use metrics in the evaluation of the quality of their software. Dr. Rosenberg's work in software metrics outside of NASA includes work with the Joint Logistics Command's efforts to establish a core set of process, product and system metrics with guidelines published in the *Practical Software Measurement*. In addition, Dr. Rosenberg worked with the Software Engineering Institute to develop a risk management course. She is now responsible for risk management training at all NASA centers, and the initiation of software risk management at NASA Goddard. As part of the SATC outreach program, Dr. Rosenberg has presented metrics/quality assurance papers and tutorials at GSFC, and IEEE and ACM local and international conferences. She also reviews for ACM, IEEE and military conferences and journals.

Immediately prior to this assignment, Dr. Rosenberg was an Assistant Professor in the Mathematics/Computer Science Department at Goucher College in Towson, MD. Her responsibilities included the development of upper level computer science courses in accordance with the recommendations of the ACM/IEEE-CS Joint Curriculum Task Force, and the advisor for computer science majors.

Dr. Rosenberg's work has encompassed many areas of Software Engineering. In addition to metrics, she has worked in the areas of hypertext, specification languages, and user interfaces. Dr. Rosenberg holds a Ph.D. in Computer Science from the University of Maryland, an M.E.S. in

Computer Science from Loyola College, and a B.S. in Mathematics from Towson State University. She is a member of Electrical and Electronic Engineers (IEEE), the IEEE Computer Society, the Association for Computing Machinery (ACM) and Upsilon Pi Epsilon.

Dr. Linda Rosenberg
GSFC
Code 300.1, Bld 6
Greenbelt, MD 20771
301-286-0087 (voice)
linda.rosenberg@gsfc.nasa.gov

Larry Hyatt

Mr. Larry Hyatt is retired from the Systems Reliability and Safety Office at NASA's Goddard Space Flight Center where he was responsible for the development of software implementation policy and requirements. He founded and led the Software Assurance Technology Center, which is dedicated to making measured improvements in software developed for GSFC and NASA.

Mr. Hyatt has over 35 years experience in software development and assurance, 25 with the government at GSFC and at NOAA. Early in his career, while with IBM Federal Systems Division, he managed the contract support staff that developed science data analysis software for GSFC space scientists. He then moved to GSFC, where he was responsible for the installation and management of the first large scale IBM System 360 at GSFC. At NOAA, he was awarded the Department of Commerce Silver Medal for his management of the development of the science ground system for the first TIROS-N Spacecraft. He then headed the Satellite Service Applications Division, which developed and implemented new uses for meteorological satellite data in weather forecasting. Moving back to NASA/GSFC, Mr. Hyatt developed GSFC's initial programs and policies in software assurance and was active in the development of similar programs for wider agency use. For this he was awarded the NASA Exceptional Service Medal in 1990.

He founded the SATC in 1992 as a center of excellence in software assurance. The SATC carries on a program of research and development in software assurance, develops software assurance guidance and standards, and assists GSFC and NASA software development projects and organizations in improving software processes and products.