International Journal of Advanced Information Science and Technology (IJAIST) ISSN: 2319:268
Vol.2, No.4, April 2013 DOI:10.15693/ijaist/2013.v2i4.24-32

Study Of Object Oriented Metrics

S.Pasupathy
Associate Professor,
Dept.of CSE, FEAT,
Annamalai University,
Tamilnadu
India.

R.Bhavani, PhD.
Professor,
Dept.of CSE, FEAT,
Annamalai University,
Tamil Nadu,
India,

ABSTRACT

This paper presents the results evaluated from our study on metrics used in object oriented software design strategies. This delivers tool-dependent metrics results and has even implications on the results of analyses based on these metrics results. The process provides a practical, systematic, start-to-finish method of selecting, designing and implementing software metrics. These metrics were evaluated using object oriented metrics tools for the purpose of analyzing quality of the product, encapsulation, inheritance, message passing, polymorphism, reusability and complexity measurement. It defines a ranking of the classes that are most vital note down and maintainability. The results can be of great assistance to quality engineers in selecting the proper set metrics for their software projects and to calculate the metrics, which was developed using a chronological object oriented life cycle process.

Key Terms: Object Oriented Software design, Software Metrics, Data Collection, Object oriented Life Cycle process

1. INTRODUCTION

In Recent years the object oriented design principles are widely used for developing good quality of product. The use of object oriented software development techniques introduces new element to software complexity measurement and set of mechanisms are used to evaluate object oriented concepts. This large variety of tools allows a user to select the tool best suited, e.g., depending on its handling, tool support, or price. However, this assumes that all metrics tools compute / interpret / implement the same metrics in the same way.

For this work, we assume that software metric (or metric in short) is a mathematical definition mapping the entities of a software system to numeric metrics values [1]. Furthermore, we understand a software metrics tool as a program which implements a set of software metrics definitions. It allows to assess a software system according to the metrics by extracting the required entities from the software and providing the corresponding metrics values. It combines software metrics values in a well-defined way to aggregated numerical values in order to aid quality analysis and assessment. As regards the research in software metrics, it has undergone a great evolution: in the first period the focus was very much on inventing new metrics for the different attributes of software, without so much regard for the scientific validity of the metrics. In recent times instead, a lot of work has been done on how to apply the theory of measurement to software metrics and how to ensure their validity.

These Metrics try to capture different aspects of software product and its process [2]. Some of the metrics also try to capture the same aspects of software e.g there are a number of metrics to measure the coupling between different classes. The remainder of these paper is structure as follows: Section 2 describes the objective of this work and specify the how to evaluate the performance. Section 3 describes various object oriented metrics. Section 4 describes the comparison results of various programs. Section 4 and Section 5 specifies the experimental results and our interpretations for the two main questions respectively. In Section 7, we discuss threats to the validity of our study. Finally, in Section 8, we conclude our findings.

2. OBJECTIVE

The objective of the paper is

1) To describe the current state-of-the-art in the measurement of software products and process.
2) Normal statistical inaccuracies can be dealt with by using multiple data sources and estimating methodologies, or by using multiple organizations to do the estimating and check and analyze results.
3) The earlier the estimate is made the less is known about the software to be developed and the greater the estimating errors.
4) To find whether each measure is independent or we can chose a subset of these metrics having equal utility as original metrics set.
5) To analyze a system performance on object oriented grounds and measure the design and code quality.
6) To cover the basic structural mechanisms of the object oriented paradigm.

3. WHY MEASUREMENT:

The main use of measurement is to evaluate quality and reuse the corresponding program into various application.

a) To calculate object oriented concepts like classes, objects, complexity(Halstead and McCabe's), encapsulation(Hiding factor), Inheritance, polymorphism, Message Passing, Coupling, Cohesion and Reuse ratio.
b) To evaluate metrics from existing measurement tools and they are commercially used for validate the performance. E.g Chidamber & Kemerer Metrics Tool, MOOD Metrics [3][4].

The attributes of entities can be internal or external.

Internal attributes of an entity can be measured only based on the entity and therefore the measures are direct. For example size is an internal attribute of any software document.

External attributes of an entity can be measured only with respect to how the entity relates with the environment and therefore can be measured only indirectly. For example, reliability, an external attribute of a program, does not depend only on the program itself but also on the compiler, machine and user [5][6]. Productivity, an external attribute of a person, clearly depends on many factors such as the kind of process and the quality of the software delivered. The entities considered in software measurement are

1. ***product*****:** any artifact produced during software development.

Table 1: Product Oriented Metrics (Entity and Attributes)

Entity	Internal Attribute	External Attribute
Requirements	Size, Reuse, Modularity, Redundancy, Functionality	Understandability, Stability
Specification	Size, Reuse, Modularity, Redundancy, Functionality	Understandability, Maintainability
Code	Size, Reuse, Modularity, Coupling, Cohesion, , Control Flow Complexity	Reliability, Usability, Reusability, Maintainability
Test set	Size, Coverage level	Quality

2. ***processes*****:** any activity related to software development.

Table 2: Process Oriented Metrics (Entity and Attributes)

Entity	Internal Attribute	External Attribute
Requirements Analysis	Time, Effort	Cost effectiveness
Specification	Time, Effort, Number of requirements changes	Cost effectiveness
Design	Time, Effort, Number of requirements changes	Cost effectiveness
Code	Time, Effort, Number of requirements changes	Cost effectiveness
Test set	Time, Effort, Number of code changes	Cost effectiveness

3. ***resource*****:** people, hardware, or software needed for the processes.

Table 3: Resource Oriented Metrics (Entity and Attributes)

Entity	Internal Attribute	External Attribute
Personnel	Age, Cost	Productivity, Experience
Team	Size, Communication Level, Structure	Productivity
Software	Size, Communication Level, Structure	Usability, Reliability
Hardware	Price, Speed, Memory size	Usability, Reliability

The external attributes are clearly the most interesting from the point of view of the manager, but they can be measured only indirectly. For example, productivity of people can be measured as the ratio of size of product delivered (an internal code attribute) and effort (an internal process attribute). Furthermore, external attributes are difficult to define: it is rare that there is a consensus on the definitions of these attributes [7][8]. For example, quality can be defined as the ratio of faults discovered during formal testing (an internal process attribute) and size, measured by KLOC (Kilo Lines Of Code)[9]. In alternative, quality can be considered as a very high-level attribute constituted by a combination of reliability, availability, maintainability and usability. In turn, maintainability comprises understandability, modifiability and testability. Moreover each of these component is influenced by complexity. So we see that external attributes are not isolated from each other but are closely related.

12 STEPS IN OBJECT ORIENTED METRICS EVALUATION

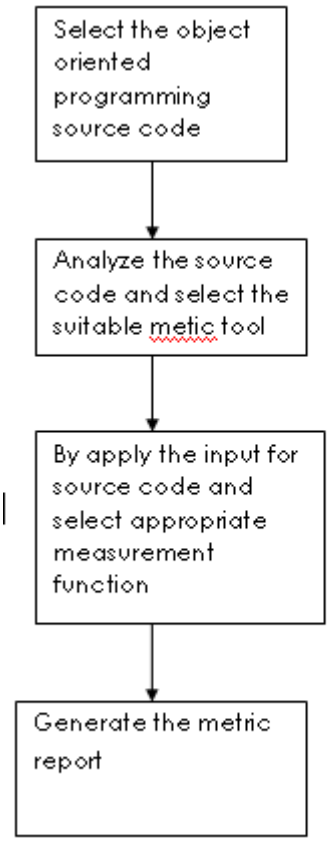


Fig .2 Flow chart for Evaluation process of Object Oriented Metrics

4. SYSTEM METRICS

The metrics are selected from most well known metrics that have been proposed and could be easily applied to object oriented programming as well [10][11]. The major system metrics are used to evaluate complexity of the program[12].

These complexity comes under time and execution of the program. These analyzed from existing tools Halstead complexity metrics suite and McCabe's Complexity metrics suite[13][14][15] our proposed work evaluate average, maximum and minimum cyclomatic complexity of the program.

5. OBJECT ORIENTED METRICS

To Measure object oriented concepts in the Software product is Object oriented Metrics.

Some of the object oriented metrics attributes are

- Number of classes
- Number of Methods
- Lines of codes
- Weighted Methods per Class
- Coupling Between Object
- Depth of Inheritance
- Number of Children
- Number of Packages
- Coupling Factor
- Reuse Ratio
- Specialization Ration
- Polymorphism factor
- Number of loop
- Number of bugs
- Method of Hiding Factor
- Attribute Hiding Factor
- Message Passing Call for Factor
- Number of Attributes per class
- Response for a class
- Lack of cohesion in method

5.1 Class Oriented Metrics

Classes, which are the central points of every object oriented language implement methods and define attributes. The class metrics address thus this aspect: their complexity can be expressed through methods and attributes and the way these entities behave. Hierarchy nesting level (HNL) also called depth of inheritance tree. The number of classes in superclass chain of class. In case of multiple inheritances, count the number of classes in the longest chain.

5.1.1 Number of Class Measurement

- NA *Number of accessors*, the number of get/set - methods in a class.
- NAM *Number of abstract methods*.
- NC *Number of constructors*.
- NCV *Number of class variables*.
- NIA *Number of inherited attributes*, the number of attributes defined in all superclasses of the subject class.
- NIV *Number of instance variables*.
- NMA *Number of methods added*, the number of methods defined in the subject class but not in its superclass.
- NME *Number of methods extended*, the number of methods redefined in subject class by invoking the same method on a superclass.

- NMI *Number of methods inherited*, i.e. defined in superclass and inherited unmodified.
- NMO *Number of methods overridden*, i.e. redefined in subject class.
- NOC *Number of immediate children of a class*.
- NOM *Number of methods*, each method counts as 1. NOM = NMA + NME + NMO.
- NOMP *Number of method protocols*. This is Smalltalk - specific: methods can be grouped into method protocols.
- PriA *Number of private attributes* and PriM *Number of private methods*.
- ProA *Number of protected attributes and* ProM *Number of protected methods*.
- PubA *Number of public attributes* and PubM *Number of public methods*.
- WLOC *Lines of code*, sum of all lines of code in all method bodies of the class.
- WMSG *Number of message sends*, sum of number of message sends in all method bodies of class. WMCX *Sum of method complexities*.
- WNAA *Number of times all attributes defined in the class are accessed*.
- WNI *Number of method invocations*, i.e. in all method bodies of all methods and WNMAA *Number of all accesses on attributes* and WNOC *Number of all descendants*, i.e. sum of all direct and indirect children of a class.
- WNOS *Number of statements*, sum of statements in all method bodies of class.

5.1.2 Methods present in the class

Methods can be seen as a flow of instructions which take input through parameters and which produce output. Methods can invoke other methods or access attributes. The method metrics are defined in this context.

- LOC *Lines of code* in method body.
- MHNL *Hierarchy nesting level* of class in which method is implemented.
- MSG *Number of message sends* in method body.
- NI *Number of invocations* of other methods in method body.
- NMAA *Number of accesses on attributes* in method body.
- NOP *Number of parameters* which the method takes.
- NOS *Number of statements* in method body.
- NTIG *Number of times invoked by methods non-local to its class*, i.e. from methods implemented in other classes.
- NTIL *Number of times invoked by methods local to its class*, i.e. from methods implemented in the same class.

5.1.3 Attributes present in the class

Attributes are properties to classes. Their main function is to return their value when accessed by methods. The attribute metrics are defined in such a context.

- AHNL *Hierarchy nesting level* of class in which attribute is defined.
- NAA *Number of times accessed.* NAA = NGA + NLA.
- NCM *Number of classes having methods that access it.*
- NGA *Number of times accessed by methods non-local* to its class.
- NLA *Number of times accessed by methods local* to its class.
- NM *Number of methods accessing it.*

5.2 Method Metrics

There are three basic methods for measuring method size. Historically, the primary measure of software size has been the number SLOC. However, it is difficult to relate software functional requirements to SLOC, especially during the early stages of development. An alternative method, function points, should be used to estimate software size. Function points a reused primarily for management information systems (MISs), whereas, feature points (similar to function points) are used for real-time or embedded systems. SLOC, function points, and feature points are valuable size estimation techniques. Fig 6 and 7 summarizes the differences between the function point and SLOC methods.

5.3 Encapsulation Metric

The encapsulation metrics evolves packaging (or binding together) of a collection of items.

- Low-level examples of encapsulation include records and arrays.
- Subprograms (e.g., procedures, functions, subroutines, and paragraphs) are mid-level mechanisms for encapsulation.
- In object-oriented (and object-based) programming languages, there are still larger encapsulating mechanisms, e.g., C++'s classes, Ada's packages, and Modula 3's modules.

Information Hiding is the suppression (or hiding) of details.

- The general idea is that we show only that information which is necessary to accomplish our immediate goals.
- There are degrees of information hiding, ranging from partially restricted visibility to total invisibility.
- Encapsulation and information hiding are not the same thing, e.g., an item can be encapsulated but may still be totally visible. Information hiding plays a direct role in such metrics as object coupling and the degree of information hiding.

5.4 Reuse Metrics

Reuse Ratio (U):

The reuse ratio (U) is given by U=number of super class/total number of class.

Specialization Ratio(S):

This ratio measures the extent to which a super class has captured abstraction. S=number of subclass/number of super class.

Average Inheritance Depth:

The inheritance structure can be measured in terms of depth of each class with in its hierarchy. Average inheritance depth = sum of depth of each class/number of class.

5.5 Quality Metrics

Reusability: Reusability means reflects the presence of OO Design characteristics that allow a design to be reapplied to new problem without significant. Reusability formula= (-0.25*coupling) + (0.25*cohesion) + (0.5*messaging) + (0.5*design size).

Flexibility: Characteristics that allow the incorporation of change in a design. The ability of a design to be adapted to provide Functionality related capabilities. Flexibility formula= (0.25*encapsulation)
(0.25*coupling)+(0.5*composition)+(0.5*polymorphism).

Understandability: The properties of the design that enable it to be easily learned and comprehend. Understandability formula= (-0.33*abstraction) + (0.33*encapsulation)-(0.33*coupling) + (0.33*cohesion)-(0.33*polymorphism)-(0.33*complexity)-(0.33*design size).

Functionality: The responsibilities assigned to the classes of design, which are made available by the classes through their public interfaces. Functionality formula= (0.12*cohesion) + (0.22*polymorphism) + (0.22*messaging)+ (0.22*design size) + (0.22*hierarchies).

Extendibility: It refers to the presence and usage of properties in an existing design that allow for the incorporation of new requirements in the design. Extendibility formula = (0.5*Abstraction)-(0.5*coupling) + (0.5*inheritance) + (0.5*polymorphism).

Effectiveness: It refers to a design's ability to achieve the desired functionality and behavior using OO Design concepts. Effectiveness formula= (0.2*abstraction) +(0.2*encapsulation) + (0.2*composition) + (0.2*inheritance) + (0.2*polymorphism).

5.6 Overall Metrics Status

A software measurement is a quantifiable dimension, attribute, or amount of any aspect of a software program, product, or process. It is the raw data which are associated with various elements of the software process and product. Metrics (or indicators) are computed from measures. They are quantifiable indices used to compare software products, processes, or projects or to predict their outcomes. With metrics, we can **Monitor** requirements, **Predict** development resources, **Track** development progress and **Understand** maintenance costs.

Table 4. Example for Selecting Measurement Tool.

AREA	MEASURES
Requirements	CSCI requirements, CSCI design stability
Performance	Input/output bus throughout, capability, Processor memory utilization, Processor throughput, utilization
Schedule	Requirements allocation status, Preliminary design status, Code and unit test status Integration status
Cost	Person-months of effort, Software size

6. PERFORMANCE MATCHING

6.1 Object Oriented Metrics in Software Engineering Approach

"Given the central role that software development plays in the delivery and application of information technology, managers are increasingly focusing on process improvement in the software development area. This demand has spurred the provision of a number of new and/or improved approaches to software development, with perhaps the most prominent being object-orientation (OO). In addition, the focus on process improvement has increased the demand for software measures, or metrics".

6.2 Applying And Interpreting Object Oriented Metrics

"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."

6.3 Design Principles and Design Patterns

"What is software architecture? The answer is multitier. At the highest level, there are the architecture patterns that define the overall shape and structure of software applications1. Down a level is the architecture that is specifically related to the purpose of the software application. Yet another level down resides the architecture of the modules and their interconnections. "

6.4 Using Educational Tools For Teaching Object Oriented Design And Programming

"The development of software systems is a complex process which requires a diverse set of skills and expertise. The Object Oriented programming paradigm has been proven to better organize the inherent complexity of software systems, than the traditional procedural paradigm. Hence, Object Oriented (OO) is becoming the dominant paradigm in the recent years. The software industry is placing increasing emphasis on newer, object-oriented programming languages and tools, such as Java. It is highly interested for software engineers capable to analyze and develop systems using the OO programming paradigm."[16]

6.5 Quality Metrics Tool For Object Oriented Programming

"Metrics measure certain properties of a software system by mapping them to numbers (or to other symbols) according to well-defined, objective measurement rules [17][18].Design Metrics are measurements of the static state of the project's design and also used for assessing the size and in some cases the quality and complexity of software. Assessing the Object Oriented Design (OOD) metrics is to predict potentially fault-prone classes and components in advances"

6.6 Message Creation Overhead and Performance

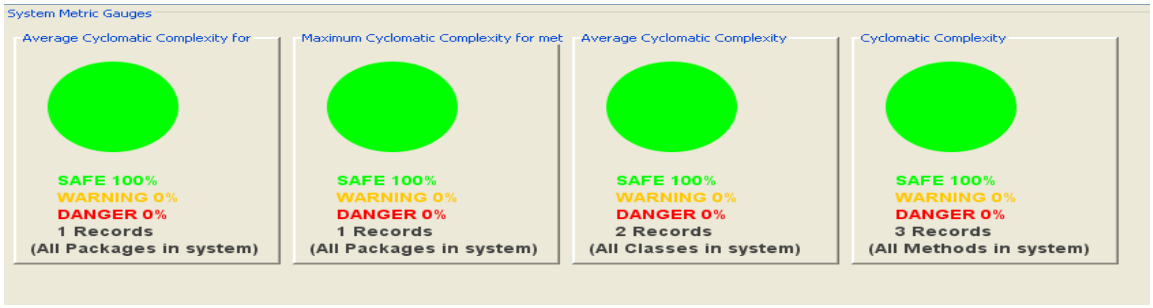
Since all messages and parameters must possess particular meanings to be consumed (i.e., result in intended logical flow within the receiver), they must be created with a particular meaning. Creating any sort of message requires overhead in either CPU or memory usage. Creating a single integer value message (which might be a reference to a string, array or data structure) requires less overhead than creating a complicated message such as a SOAP message. Longer messages require more CPU and memory to produce. To optimize runtime performance, message length must be minimized and message meaning must be maximized.

7. Experimental Result:

By applying the following code to Jhawk Metric tool, we got the following result.

```
package src;

class Sample

{

int a,b,c;

public Sample()

{

}

public void dis()

{

}

}

public class Demo extends Sample

{

int a=10;

int b=20;

public static void main(String args[])

{

Sample ob = new Sample();

}

}
```

Piechart For System Metrics:



Overall Metrics

Name of system		No System Name	
Total number of Packages	1	**Total number of Java statements**	10
Average Cyclomatic Complexity of the meth...	0.67	**Cumulative Halstead bugs**	0.06
Cumulative Halstead effort	604.51	**Maintainability Index**	135.15
Total Number of Comment Lines in the Syst...	0	**Total Lines of Code in the System**	20
Total number of Classes	2	**Total Number of Comments in the system**	0
Cumulative Halstead length	57	**Total number of methods**	3
Total Cyclomatic Complexity	2	**Maintainability Index (Not including commen...**	135.15
Cumulative Halstead volume	185.59		

Packages

Name	No. Classes	NOS	AVCC	HBUG	HEFF	MI	CCML	NLOC
src	2	10	0.67	0.06	604.51	135.15	0	2

Method Metrics

All Methods In System

Name	COMP	NOCL	NOS	HLTH	HVOC	HEFF	HBUG	CREF	XMET	LMET	NLOC
Sample() (src...	1	0	1	4	4	12.00	0.00	0	0	0	
dis() (src.Sa...	1	0	1	5	5	17.41	0.00	0	0	0	
main(java.la...	1	0	2	16	14	284.28	0.02	2	1	0	

Halstead length

Class Metrics

All classes in system

Name	No. Me...	LCOM	AVCC	NOS	HBUG	HEFF	UWCS	INST	PACK	RFC	CBO	MI	CCML	NLOC
Demo	1	2.00	1.00	5	0.04	525.34	3	2	0	2	1	123.50	0	
Sample	2	3.00	1.00	4	0.02	74.41	5	3	0	2	1	147.40	0	

8. CONCLUSION AND FUTURE SCOPE

The above results can be used in order to determine when and how each of the above metrics can be used according to quality characteristics a practitioner wants to emphasize. Make sure the software quality metrics and indicators they employ include a clear definition of component parts are accurate and readily collectible, and span the development spectrum and functional activities. Survey data indicates that most organizations are on the right track to making use of metrics in software projects. For organizations which do not reflect "best practices", and would like to enhance their metrics capabilities, the following recommendations are suggested to Measure the "best practices" list of metrics more consistently across all projects. Focus on "easy to implement" metrics that are understood by both management and software developers, and provide demonstrated insight into software project activities[19].

A number of object oriented metrics have been proposed in the literature for measuring the design attributes such as inheritance, polymorphism, message passing, complexity, Hiding Factor, coupling, cohesion, reusability etc,[19][20]. The number of methods and the complexity of methods involved is a predictor of how much time and effort is required to develop and maintain the class. This metrics set can be applied on various projects and evaluate and compare the performance of the code using object oriented paradigm. While in the past the focus in research was on inventing new metrics, now the focus is more on measurement theory, in particular on the definition of new validation frameworks or of new set of axioms. A practical, systematic, start-to-finish method of selecting, designing, and implementing software metrics is a valuable aid[21].

8. REFERENCES

[1] J. Alghamdi, R. Rufai, and S. Khan. Oometer: A software quality assurance tool. Software Maintenance and Reengineering, 2009. CSMR 2009. 9th European Conference on, pages 190{191, 21-23}, March 2010.

[2] H. Bsar, M. Bauer, O. Ciupke, S. Demeyer, S. Ducasse, M. Lanza, R. Marinescu, R. Nebbe, O. Nierstrasz, M. Przybilski, T. Richner, M. Rieger, C. Riva, A. Sassen, B. Schulz, P. Steyaert, S. Tichelaar, and J. Weisbrod. The FAMOOS Object-Oriented Reengineering Handbook, Oct. 2006.

[3] B. Bohem, Software Engineering Economics, Prentice Hall, Englewood Cliffs, 1981 [Briand et al 94] L. Briand, S. Morasca, V. Basili, Defining and Validating High-Level Design Metrics, Tech. Rep. CS TR-3301, University of Maryland, 2009.

[4] L. Briand, S. Morasca, V. Basili, Property-Based Software Engineering Measurement, IEEE Trans. Software Eng. 22(1), 2000, pp. 68-85.

[5] Kaur Amandeep, Singh Satwinder, K. Kahl. "Evaluation and Metrication of Object Oriented System", International Multi Conference of Engineers and Cmputer Scientists, 2009 vol. 1.

[6] M. Xenos, D.Stavrinoudis, K.Zikouli and D. Christodoulakis, "Object Oriented Metrics – A Survey", Proceeding of the FESMA 2000, Federation of European Software Measurement Association, Madrid. Spain, 2006.

[7] V. Basili, Qualitative Software Complexity Models: a Summary, in Tutorial on Models and Methods for Software Management and Engineering, IEEE Computer Society Press, Los Alamitos, CA, 2004.

[8] A. Albrecht: "Measuring application development productivity", in Proc. Joint SHARE/GUIDE/IBM Applications Development Symposium, Monterey, CA, 2007.

[9] A. Albrecht and J. Gaffney: Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation; in IEEE Trans. Software Eng., 9(6), 2008, pp. 639-648.

[10] S. Conte, H. Dunsmore, V. Shen, Software Engineering Metrics and Models, Benjamin/Cummings, Menlo Park, CA.

[11] S. Chidamber, C. Kemerer, A Metrics Suite for Object Oriented Design, IEEE Trans. Software Eng., 20/6), 2000, pp. 263-265.

[12] S. Morasca, Software Measurement: State of the Art and Related Issues, slides from the School of the Italian Group of Informatics Engineering, Rovereto, Italy, September 2008.

[13] J. Stathis, D. Jeffrey, An Empirical Study of Albrecht's Function Points, in Measurement for Improved IT management, Proc. First Australian Conference on Software Metrics, ACOSM 93, Sydney, 2002, pp. 96 - 117.

[14] E. Weyuker, Evaluating Software Complexity Measures, IEEE Trans. Software Eng., 14(9), 2002, pp. 1357-1365.

[15] H. Zuse, Software Complexity: Measures and Methods, Walter de Gruyter, Berlin, 2006.

[16] Ada and C++: A Business Case Analysis, Office of the Deputy Assistant Secretary of the Air Force, Washington, DC, June 1999.

[17] Albrecht, A.J., "Measuring Application Development Productivity," Proceedings of the IBM Applications Development Symposium, Monterey, California, October 2005.

[18] Boehm, Barry W., Software Engineering Economics, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 2006.

[19] Boehm, Barry W., as quoted by Ware Myers, "Software Pivotal to Strategic Defense," IEEE Computer, January 2001.

[20] Campbell, Luke and Brian Koster, "Software Metrics: Adding Engineering Rigor to a Currently Ephemeral Process," briefing presented to the McGrummwell F/A-24 CDR course, 2003.

[21] Carey, Dave and Don Freeman, "Quality Measurements in Software," G. Gordon Schulmeyer and James I. McManus, eds., Total Quality Management for Software, Van Nostrand Reinhold, New York, 2005.