ISSN 2277-3061

# EVALUATION OF SOFTWARE METRICS FOR SOFTWARE PROJECTS

Kunal Chopra [1], Monika Sachdeva [2]
[1] Research Scholar, Department of Computer Science Engineering, SBSSTC, Ferozepur
kunalraichopra@gmail.com
[2] Associate Professor, Department of Computer Science Engineering, SBSSTC, Ferozepur
monika.sal@rediffmail.com

## ABSTRACT

Software metrics are developed and used by the many software organizations for the evaluation and confirmation of good code, working and maintenance of the software product. Software metrics measure and identify various types of software complexities such as size metrics, control flow metrics and data flow metrics. One of the significant objective of software metrics is that it is applicable to both a process and product metrics. Ndepend is the most advanced as well as flexible tool available in the market. We have ensured the Quality of the project by using Ndepend metrics. So we have concluded that software metrics are easy to understand and applicable on the software, so favourable among software professionals.It is most prevalent and important testing metrics used in organizations. Metrics are used to improve software productivity and quality. This thesis introduces the most commonly used software metrics proposed and reviews their use in constructing models of the software development process.

## Keywords

Software metrics, Quality of service, Software Engineering.

ISSN 2277-3061

## INTRODUCTION

Software metric is a measurement derived from a software product, process, or resource. Its purpose is to provide a quantitative assessment of the extent to which the product, process, or resource possesses certain attributes. According to Costars [2] software metrics are necessary in order to electively manage software development. This can be done by accurate schedule, better quality products, good productivity and good cost estimates. The goals of software metrics are to identify and measure essential factors which effects software development. However, organizations face certain hindrances for metrics such as ill defined metrics, misinterpretation of software life cycle, incompleteness of models, weakness of measurement atomization and lack of metric validation. Metrics that are gathered from requirements phase include size metrics constituting functions, lines of code and complexity. Quality of requirements for example, can be measured as volatility - The degree to which requirements changes over period of time.

Traceability can be from requirements to requirements and requirements to design or test documents. Consistency and Complexity can also be gathered from requirements phase. “Metrics don’t solve problem, people solve problem, and Metrics provide information so that people can solve problems.” Metrics was introduced in the context of software measurement which has become an essential tool for the good software engineering. So Metric can be defined as a quantitative measure of degree to which a system, component or process possesses a given attribute. “A handle or guess about a given attribute” for example “Number of errors found per person hours expended.”

Measurement is needed at least for accessing the status of the projects, process, products and resources. Because we do not always know what details a project, it is essential that we measure and record characteristics of good project as well as bad. “You cannot control what you cannot measure.”

Every measurement action must be motivated by a particular goal or need that is clearly defined and easily defined. Now we quote few examples of the kind of the information needed to control and understand a software development project, separated by manager and developer perspective:

• We can measure time and effort involved in the various processes that comprises software production. For example, we can identify the cost of eliciting requirements, the cost of specifying the system, the cost of designing the system, and the cost of coding and testing the code.

• We can measure the time it takes for staff to specify the system, design it, code it and test it. By determining these factors we can predict how productive the staff is at each activity.

## RELATED WORK

Most of researchers have been working in the area of recognizing and evaluating the software metrics to ensure the QOS. So as to appreciate their contribution and better understand the work ahead.

**Wohlin** et al. [8] Software properties and the effect of introducing new methods and tools have to be measured. We need to manage all aspects of software (e.g. structure, resource allocation, qualities) in an optimal way. This is not possible without the aid of software metric. This paper will discuss the need of software metrics, classify qualities, and present some explicit results from research into complexity, reliability and correctness.

**Ming- Chang Lee** et al. [9] Software measurement process must be a good oriented methodical process that measures, evaluates, adjusts, and finally improves the software development process. The main contribution of this work is the easy and extensible solution to software quality of validation and verification In software evelop process. Therefore, we use formal approaches in order to describe the fundamental aspects of the software.

**Majdi Abdellatief** et al. [10] The motivation of this paper is that the measurement based on the flow of information connecting software components can be used to evaluate component-based software system dependency. The ability to measure system dependency implies the capability to locate weakness in the system design and to determine the level of software quality. In this paper, dependency between components is considered as a major factor affecting the structural design of Component-based software System (CBSS).

**Dr. Linda Rosenberg** et al. [11] The IEEE defines reliability as “The ability of a system or component to perform its required functions under stated conditions for a specified period of time.” To most project and software development managers, reliability is equated to correctness, that is, they look to testing and the number of “bugs” found and fixed. While finding and fixing bugs discovered in testing is necessary to assure reliability, a better way is to develop a robust, high quality product through all of the stages of the software lifecycle.

**Javaid Iqbal** et al. [12] Reliability of a software system has been one of the driving forces for the various software engineering processes and methodologies that led to their evolution and sophistication. The concerns for reliability of a software system surface very early on during the development phases of the software system. When it comes to acquisition of reliability, we should not immediately get model-oriented; instead every minutia of software development life cycle should be given its due. This paper outlines the areas where reliability needs to pick up.

**Rajiv D.Banker and Robert J. Kauffrnan** et al. [13] studied that automated collection of software metrics in computer aided software engineering (CASE) environments opens up new avenues for improving the management of software development operations, as well as shifting the focus of management's control efforts from "software project to "software assets" stored in a centralized repository. Repository evaluation, a new direction for software metrics research in the 1990s,promises a fresh view of software development performance for a range of responsibility levels. We discuss the

automation of function point and code reuse analysis in the context of an integrated CASE (I-CASE) environment deployed at a large investment bank in New York City.

**Tore Dybå**, et al.[14] Empirical studies of agile software development: A systematic review Agile software development represents a major departure from traditional, plan-based approaches to software engineering. A systematic review of empirical studies of agile software development up to and including 2005 was conducted. The search strategy identified 1996 studies, of which 36 were identified as empirical studies. The studies were grouped into four themes: introduction and adoption, human and social factors, perceptions on agile methods, and comparative studies.

**Matthew James Munro** et al. [15] Refactoring can have a direct influence on reducing the cost of software maintenance through changing the internal structure of the source-code to improve the overall design that helps the present and future programmers evolve and understand a system. Bad smells are a set of design problems with refactoring identified as a solution. Locating these bad smells has been described as more a human intuition than an exact science. This paper addresses the issue of identifying the characteristics of a bad smell through the use of a set of software metrics

**Gurdev Singh** et al. [16] proposed study of software metrics. He studied and evaluated various software metrics which are used in different software development process. Poor size estimation is one of the main reasons major software-intensive acquisition programs ultimately fail. Size is the critical factor in determining cost, schedule, and effort. The failure to accurately predict (usually too small) results in budget overruns and late deliveries which undermine confidence and erode support for your program. Size estimation is a complicated activity, the results of which must be constantly updated with actual counts throughout the life cycle. Size measures include source lines-of-code, function points, and feature points.

**Shahid Iqbal** et al. [17] Proposed metrics to ensure quality in Requirement Engineering Process. He also discussed the effects of the proposed metrics in Requirement Engineering process. Software project management has emerged as a new discipline with wide-ranging ideas and across-the-board insights for effectively managing key areas of software projects. The remarkable work of the software project managers, professionals and researchers across the globe have resulted in substantial improvements in this field. Likewise, software project failure rate has decreased considerably due to the use of effective software project management tools and techniques by the software houses. Now more efficient, robust and quantitative measures are being practiced in the areas of software requirement gathering, analysis, design, architecture, development, quality assurance, integration, deployment and support.

**Mrinal Singh Rawat** et al.[18] identified different object oriented metrics and models to improve the software quality and also discussed impact of software metrics in software quality. Software metrics provide a quantitative basis for planning and predicting software development processes. Therefore the quality of software can be controlled and improved easily. Quality in fact aids higher productivity, which has brought software metrics to the forefront. This research paper focuses on different views on software quality.

**Vikas Verma** et al. [19] The various software testing metrics and models and to meet out the objectives this paper is confined to the Software Testing companies. The developers of software companies who are working as software testers participated in the study. The metrics relevant to process improvement can be effectively identified and tailored to the organization and its goals and to ensure consistency and completeness, measurement provides the most appropriate information. Metrics are the most important responsibility of the Test Team. Metrics allow for deeper understanding of the performance of the application and its behaviour. This paper introduces the most commonly used software metrics proposed and reviews their use in constructing models of the software development process.

## RESEARCH GAP

The given problem for this thesis has two broader parts. The first part, research, is about the study, exploration and categorization of software metrics. The second part is to ensure the quality of a solution for achieving automatic code complexity measurement using software metrics. An organization needs continuous improvement to its software process in order to remain competitive in the market. It should assess its functions and how it is achieving its goals. In the IT industry, we will never apply the same set of metrics for all projects because each project will carry its own requirements, methodology and logic. So we need to redefine the categories of the metrics according to projects taken by the company. Until or unless, we don't apply the metric we can never be assured about the quality of the project. So our focus will be on categorization of metrics and on implementing the metrics using Ndepend tool on different types of software projects.

## OBJECTIVES

- To select Software Metrics for ensuring the quality of software projects.
- Categorization of software metrics into Size Based Metrics, Dependency Metrics, Inheritance Based Metrics, Metrics showing Project Readability and Metrics stating Project Objectivity.
- Selection of simulator for evaluating the software metrics.
- Computation of selected metrics with the help of selected Simulator.
- Comparison of computed metrics with the Threshold values of Metrics
- If the results are deviated from the threshold values, then we will try to improve the module whose results are not in the given range.

ISSN 2277-3061

• Re-compute the software metrics and keep on improving the module until the results are satisfied.

• Improve the quality of software projects based on results of computed Metrics with the help of Ndepend Simulator.

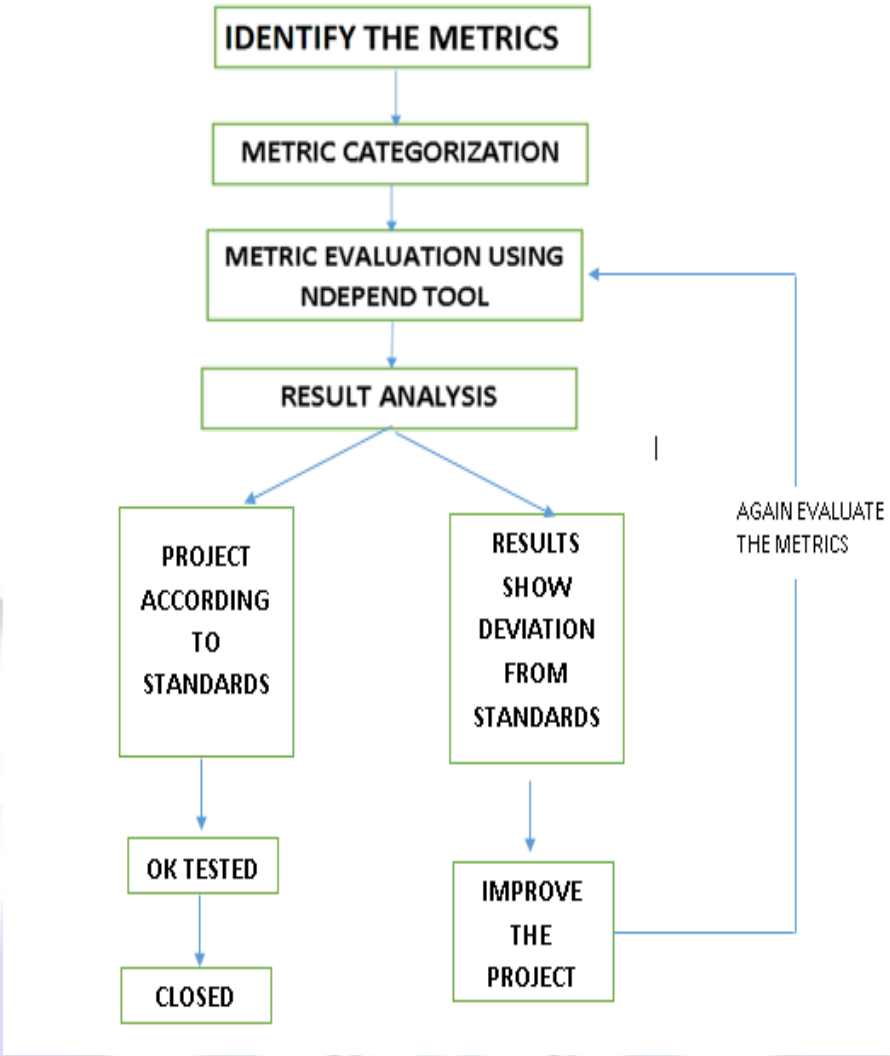


**Figure 1. Methodology of the proposed work**

## METHODOLOGY

In this section, we present the way to identify the varied options of metrics as if the huge range of metrics are available for measuring the software therefore we need to derive the methodology to identify the required metrics in accordance with our project in hand. So in that case our focus is on to categorize the metrics into so as to get reliability for the selection of metrics amongst the varied range of metrics:

1. SIZE BASED METRICS
2. DEPENDENCY METRICS
3. INHERITANCE BASED METRICS
4. METRIC SHOWING PROJECT READABILITY
5. METRIC STATING PROJEC OBJECTIVITY

Now based on the project code in what platform or environment the project has been created the selection of simulator is required to evaluate or calculate the selected metrics for the process improvement. After evaluating the metrics we need to compare each calculated value of metric with the threshold value of the metric. If the results are deviated from the threshold values, then we will try to improve the module whose results are not in the given range. Re-compute the software metrics and keep on improving the module until the results are satisfied. Likewise we improve the quality of software projects based on results of computed Metrics with the help of Ndepend Simulator.

ISSN 2277-3061

## NDEPEND

NDepend is a static analysis tool for .NET managed code. This tool supports a large number of code metrics, allows for visualization of dependencies using directed graphs and dependency matrix. The tools also performs code base snapshots comparison, and validation of architectural and quality rules. User-defined rules can be written using LINQ queries. This possibility is named CQLinq. The tool also comes with a large number of predefined CQLinq code rules. Code rules can be checked automatically in Visual Studio or during continuous integration.

The main features of NDepend are:

• Dependency Visualization (using dependency graphs, and dependency matrix)

• Software metrics (NDepend currently supports 82 code metrics: Cyclomatic complexity; Afferent and Efferent Coupling; Relational Cohesion; Google page rank of .NET types; Percentage of code covered by tests, etc.)

• Declarative code rule over LINQ query (CQLinq)

• Integration with CruiseControl and TeamCity

• Optional code constraints in the source code using CLI attributes

• Version comparison of two versions of the same assembly.

## SELECTED METRICS

We have arranged the metrics into five different categories and the metrics under these categories are:

**1. Size Based Metrics:**

- LOC
- NbMethods
- ILInstruction

**2. Dependency Metrics**:

- AFFERENT COUPLING
- EFFERENT COUPLING

**3. Inheritence Based Metrics**

- DIT (Depth Of Inheritence)
- NOC (Number Of Children)

**4. METRIC SHOWING PROJECT READABILITY**

- COMMENT PERCENTAGE

**5. METRIC STATING PROJECT OBJECTIVITY**

- LCOM – LACK OF COHESION OVER METHODS

**LOC (NbLinesOfCode)**: (defined for application, assemblies, namespaces, types, methods) This metric (known as LOC) can be computed only if PDB files are present. NDepend computes this metric directly from the info provided in PDB files.

The LOC for a method is equals to the number of sequence point found for this method in the PDB file. A sequence point is used to mark a spot in the IL code that corresponds to a specific location in the original source. Computing the number of lines of code from PDB's sequence points allows to obtain a logical LOC of code instead of a physical LOC (i.e. directly computed from source files). 2 significant advantages of logical LOC over physical LOC are:

**PercentageComment:** (defined for application, assemblies, namespaces, types, methods) PercentageComment = 100 * NbLinesOfComment / ( NbLinesOfComment + NbLinesOfCode)

**NbILInstructions**: (defined for application, assemblies, namespaces, types, methods) Notice that the number of IL instructions can vary depending if your assemblies are compiled in debug or in release mode. Indeed compiler's optimizations can modify the number of IL instructions. For example a compiler can add some nop IL instructions in debug mode to handle Edit and Continue and to allow attach an IL instruction to a curly brace. Notice that IL instructions of third-party assemblies are not taken account.

**NbMethods:** (defined for application, assemblies, namespaces, types) the number of methods. A method can be an abstract, virtual or non-virtual method, a method declared in an interface, a constructor, a class constructor, a finalizer, a property/indexer getter or setter, an event adder or remover. Methods declared in third-party assemblies are not taken account.

**Afferent coupling (Ca)**: The number of types outside this assembly that depend on types within this assembly. High afferent coupling indicates that the concerned assemblies have many responsibilities.

**Efferent coupling (Ce)**: The number of types outside this assembly used by child types of this assembly. High efferent coupling indicates that the concerned assembly is dependant. Notice that types declared in third-party assemblies are taken into account.

**Lack of Cohesion Of Methods (LCOM):** The single responsibility principle states that a class should not have more than one reason to change. Such a class is said to be cohesive. A high LCOM value generally pinpoints a poorly cohesive class. There are several LCOM metrics. The LCOM takes its values in the range [0-1]. The LCOM HS (HS stands for Henderson-Sellers) takes its values in the range [0-2].

**Number of Children (NOC):** The number of children for a class is the number of sub-classes (whatever their positions in the sub branch of the inheritance tree). The number of children for an interface is the number of types that implement it. In both cases the computation of this metric only count types declared in the application code and thus, doesn't take account of types declared in third-party assemblies.

**Depth of Inheritance Tree (DIT):** The Depth of Inheritance Tree for a class or a structure is its number of base classes (including theSystem.Object class thus DIT >= 1). Types where DepthOfInheritance is higher or equal than 6 might be hard to maintain. However it is not a rule since sometimes your classes might inherit from third-party classes which have a high value for depth of inheritance.

## EXPERIMENTS

We have considered 3 projects having different modules and different complexities.

- **PROJECT 1** : This project includes the code for connecting the dotnet code with the database and for performing various operations like save, update, delete and selection of records from the database.
- **PROJECT 2** : This project contains all those libraries that are required for implementing the AJAX (Asynchronous JavaScript and XML). By using the AJAX, we can receive the data from the server without refreshing the entire web page.
- **PROJECT 3** : It is an advanced, low level C# library that is suitable for games, scientific applications and any other project that requires the graphics and animations.

Table 1,2,3,4,5 shows the computed metrics for the different projects using NDepend. These tables specifies that the metrics value will increase as the complexity of the project increases.

| METRIC | PROJECT 1 | PROJECT 2 | PROJECT 3 |
|---|---|---|---|
| LOC | 77 | 7633 | 183206 |
| NBMETHODS | 4 | 368 | 7658 |
| ILINSTRUCTION | 463 | 54776 | 1348870 |

**Table 1. Size Based Metrics Evaluation**

| METRIC | PROJECT 1 | PROJECT 2 | PROJECT 3 |
|---|---|---|---|
| AFFERENT COUPLING | 3 | 10 | 197 |
| EFFERENT COUPLING | 38 | 103 | 1137 |

**Table 2. Dependency Metrics Evaluation**

| METRIC | PROJECT 1 | PROJECT 2 | PROJECT 3 |
|---|---|---|---|
| DIT | 4 | 41 | 536 |
| NOC | 1 | 354 | 4558 |

**Table 3. Inheritance Based Metric Evaluation**

| METRIC | PROJECT 1 | PROJECT 2 | PROJECT 3 |
|---|---|---|---|
| COMMENT % | 3 | 2 | 12 |

**Table 4. Project Readability Metric Evaluation**

| METRIC | PROJECT 1 | PROJECT 2 | PROJECT 3 |
|---|---|---|---|
| LCOM | .78 | .63 | .92 |

**Table 5. Project Objectivity Metric Evaluation**

Table 6 specifies the threshold values given by the Ndepend simulator.

| METRIC | THRESHOLD VALUE |
|---|---|
| LOC | GREATER THAN 20 PER METHOD; COMPLEXITY INCREASES |
| NBMETHOD | GREATER THAN 20; COMPLEXITY INCREASES |
| NBILINSTRUCTIONS | GREATER THAN 100 PER METHOD; COMPLEXITY INCREASES |
| AFFERENT COUPLING | LOWER VALUE IS PREFERRED |
| EFFERENT COUPLING | LOWER VALUE IS PREFERRED |
| DIT | GREATER THAN 6; COMPLEXITY INCREASES |
| NOC | GREATER THAN 4; COMPLEXITY INCREASES |
| COMMENT PERCENTAGE | 20% - 40% |
| LCOM | GREATER THAN 0.8 IS PREFERRED |

**Table 6. Metric Threshold Value**

According to the threshold values given by the Ndepend, our projects are deviating from those values. So using those computed metrics, we have tried to improve the quality of the project 1.

| METRIC | Old Values | New Values |
|---|---|---|
| LOC | 77 | 52 |
| NBMETHOD | 4 | 4 |
| NBILINSTRUCTIONS | 463 | 356 |
| AFFERENT COUPLING | 3 | 3 |
| EFFERENT COUPLING | 38 | 100 |
| DIT | 4 | 4 |
| NOC | 1 | 1 |
| COMMENT PERCENTAGE | 3 | 12 |
| LCOM | .78 | .78 |

**Table 7. Improved metric values compared with the old values.**

ISSN 2277-3061

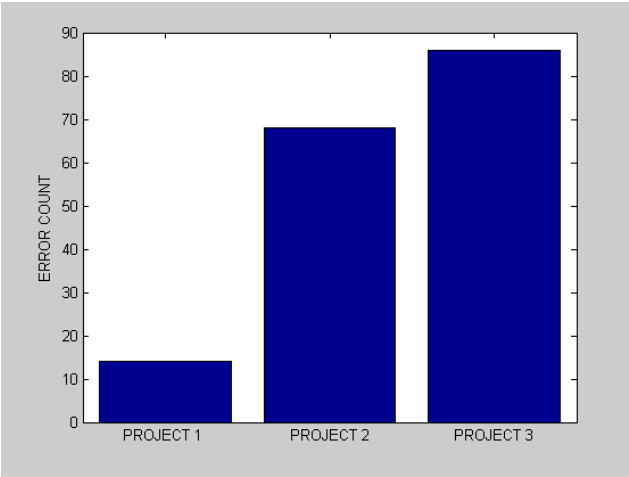


**Figure 2. Bar Chart Specifying the Error Count of Different Projects using NDepend**

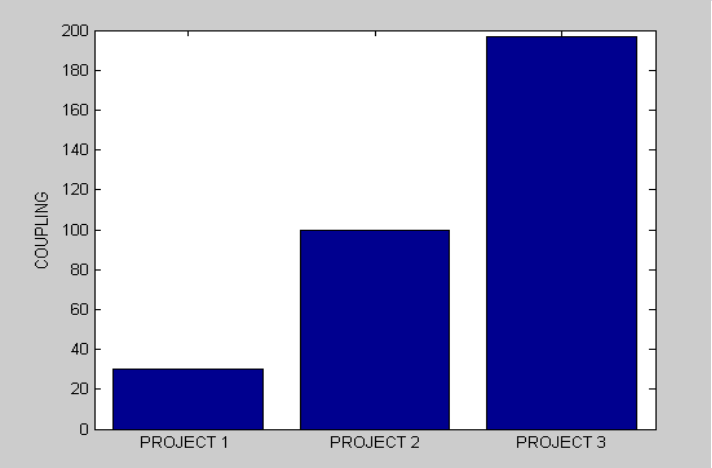


**Figure 3. Coupling Count in NDepend**

## CONCLUSION

The outcome of the developed tool shows that the tool is capable of performing code analysis automatically on regular basis. It proves that the automatic measurement of source code complexity is possible to implement. This tool can be helpful for developers to view the quality of their code in terms of code metrics. Potentially fault-prone code can easily be identified which can suggest developers about the code that requires refactoring. Comparison between projects helps the managers to observe the change of code between the releases. It is also possible to identify what parts of code have changed and how much they are changed. The change of code may also help the testers to focus their testing efforts on those parts of code that are changed. If there are new source code files added, the McCabe cyclomatic complexity metric can still be useful for the testers to know how much test cases they need to develop. The automatic logging feature allow for real-time monitoring of the tool. The gathered metrics should more thoroughly be evaluated to find possible correlations between certain metrics. The results of the gathered metrics should thoroughly be verified using other metrics gathering sources/tools. The intermediate tool used for metrics gathering can be omitted and tool can be fully developed within the company, and not to rely on a third party tool where the functionality of the tool can change for its newer versions. The graphs used for showing the comparisons are generated using Google Visualization API. It requires internet connection to be able to generate the visualizations. It can be security risk for the sensitive data. An alternate approach should be adopted where visualizations are possible even without the internet connection.

## REFERENCES

[1] DeMarco, Tom, "Controlling Software Projects", Yourdon Press, New York, 1986

[2] 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, 1995.

[3] Fenton, N., and Pfleeger, S. L. "Software Metrics - A Rigorous and Practical Approach", 2 ed. International Thomson Computer Press, London, 1996.

[4] Sam Miller, "Areas of Use for Software Metrics", 2008 [Online: accessed on 2010-06-16 from http://www.articlesbase.com/management-articles/areas-of-use-for-software-metrics306127.html

[5] Anton Milutin, "Software code metrics", 2009 [Online: accessed on 2010-06-21 from http://www.viva64.com/content/articles/codeanalyzers/?f=Metrics.html&lang=en&content=code-analyzers]

[6] Everald E. Mills, "Software Metrics", Software Engineering Institute, 1988.

[7] Lou Marco, "Measuring software complexity", Enterprise Systems Journal, 1997.

[8] S. Henry, D. Kafura, K. Mayo, A. Yerneni, S. Wake, "A Reliability Model Incorporating Software Quality Factors", Department of Computer Science, Virginia Tech, Blacksburg.

[9] Jeremy Singer, Christos Trortjis, and Martin Ward, "Using Software Metrics to Evaluate Static Single Assignment Form in GCC", University of Manchester - UK, University of Ioannina - Greece, University of Western Macedonia - Greece.

[10] Jerry Fitzpatrick, "Applying the ABC Metric to C, C++, and Java", C++ Report, June 1997.

[11] MSDN, "Code Metrics Values", [Online: accessed on 2010-06-21 from http://msdn.microsoft.com/en-us/library/bb385914.aspx ]

[12] Kurt D. Welker, Paul W. Oman, Gerald G. Atkinson, "Software Maintenance: Research and Practice Vol 9", John Wiley & Sons, Ltd, 1997.

[13] Young Lee, "Automated Source Code Measurement Environment For Software Quality", Auburn University Alabama, December 2007.

[14] M Squared Technologies, "Source Code Size Metrics", [Online: accessed on 2010-12-10 from http://msquaredtechnologies.com

[15] T. J. McCabe, “A Complexity Measure,” ICSE '76: Proceedings of the 2nd international conference on Software engineering, 1976.

[16] Complexity Metric Control, [Online: accessed on 2010-20-05 from http://gcc.gnu.org/onlinedocs/ gnat_ugn_unw /Complexity-Metrics-Control.html

[17] S. C. Johnson, "Portable C Compiler", [Online: accessed on 2010-20-05 from http://pcc.ludd.ltu.se/ ]

[18] Armin Krusko, "Complexity Analysis of Real Time Software", Royal Institute of Technology Sweden, 2002.

[19] Harrison, W., K. Magel, R. Kluczny, and A. DeKock, “Applying Software Complexity Metrics to Program Maintenance.” Computer, 1982.

[20] Rafa E. AL QUTAISH, Alain ABRAN, "An Analysis of the Design and Definitions of Halstead’s Metrics", École de Technologie Supérieure, Canada.

[21] Hamer, P. G. and Frewin, G. D., "M. H. Halstead's Software Science - A Critical Examination", in the Proceedings of the 6th International Conference on Software Engineering, Tokyo, Japan, 1982.