AUTOMATED CODE ANALYSER

Hansini M. Fernando, Damith R. Kothalawala, Dilshan I. De Silva, Nuwan Kodagoda
Sri Lanka Institute of Information Technology
New Kandy Road, Malabe, Sri Lanka
madushahansi@gmail.com, admin@drklk.org, dilshan.i@sliit.lk, nuwank@sliit.lk

ABSTRACT
Automated Code Analyzer is a software tool which is capable of providing an automated software metrics support for its users. It evaluates the quality of source codes and formulates schedule estimations according to a specified hierarchical metrics based model. It also generates Quality notices, analyses the results of metrics calculations, presents the system observations on a particular code and provide guidelines to resolve the problems observed by the system. The proposed software was designed based on four core metrics categories, namely Object Oriented, Complexity Oriented, Size Oriented and Maintainability Oriented. Unlike the existing metric calculators, this tool is capable of diminishing the problems that exists in the field of software engineering in addition to providing valuable information of a given source code using metrics.

KEY WORDS
Automated Code Analyser, Software Metrics, Object Oriented, Size Oriented, Complexity Oriented.

1. Introduction

With the rapid growth of software applications a significant concern is now placed on software quality measurements and estimations. Thus, the development of automated software tools has risen up by a considerable amount. Tom DeMacro, a famous supporter of the need for measurement in software development has stated that “You cannot control what you cannot measure” [1], [2]. This insists the need of a proper code analysing tool to evaluate the quality of software.

At present there is no well-structured model to evaluate software source code that is extremely predictive and reliable. Although there exist many code analysing tools to assist the software measurement in the industry, most of them only collect unstructured metrics. They do not provide a well-defined approach to relate the metrics to the external quality attributes of the software, which is one of the most essential concerns of the metrics users.

Following are some of the most widely used code analysing tools in the industry.

- Visual Studio Code Metrics Power Tool.
- Eclipse Metrics plug-in.
- Resource Standard Metrics.
- NDepend.

After a thorough analysis of the currently available software measurement models many limitations that exist within them were identified: lack of consistency and reliability, complexity of the measurements, unstructured organization of the metrics and less user interaction [3]. It was observed that there is still a need for a well structured software analysis tool which can handle the above mentioned problems and which is able to provide a well structured framework to analyse and comprehend the source code of software. Thus, Automated Code Analyser (ACA) was developed to address the limitations of the existing tools and thereby provide an automated software metrics support for its users.

The tool analyses the source code of a program and thus evaluates the quality of the software and formulates metrics based quality notices and estimations according to a specified hierarchical metrics model [4].

One of the unique characteristics of ACA is that it assesses the Source code based on four major metric categories namely:

- Object Oriented Metrics.
- Complexity Metrics.
- Size Metrics.
- Maintainability Metrics.

Our extreme attempt was to construct a precise code analysing and estimation scheme which would have a higher explanatory power and reliability than that of current models. The idea was not just to design a simple metric calculator but to come up with an innovative solution to diminish the problems currently existing in the region of software engineering. We have successfully achieved this goal by means of the implementation of Automated Code Analyser.

2. Methodology

To design a high-quality and well structured software it is necessary to comply with the industry standards. IEEE standards were recognized as the best set of standards to follow, while designing and developing the Code Analyser.

Software quality engineering has been defined as the formal management of quality throughout the software product lifecycle. To best support this requirement, any supporting methodology must be able to simulate the definition of the quality requirements, reporting activities, code evaluation process, and the subsequent improvements of software quality.

IEEE Standard 1061 satisfies all these requirements and provides a very convenient process of modeling Software Quality engineering practices.

The details for the approaches provided in this section are provided in IEEE Standard 1061; IEEE Standard for a Software Quality Metrics Methodology [5]. It presents three frameworks that support software metrics and measurement activities.

- The Software Quality Metrics Methodology.
- The Goal, Question, Metric (GQM) Paradigm.
- Practical Software Measurement (PSM).

Out of these three the “Software Quality Metrics Methodology” was selected as the base of this project.

2.1 Software Quality Metrics Methodology

The Software quality metrics framework presented by the IEEE Standard 1061 as shown in Figure 1 is hierarchical structure and is designed to be flexible. It begins with the establishment of quality requirements, which are used to describe the quality of software (or a system). These requirements are assigned quality attributes. It permits additions, deletions, and modifications of quality factors, quality sub factors, and metrics. Each level may be expanded to several sublevels. The framework can thus be applied to all systems and can be adapted as appropriate without changing the basic concept [5].

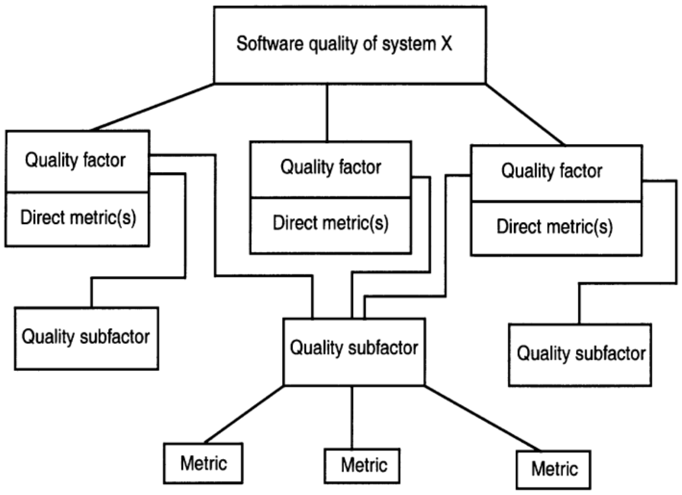


Figure 1. Software quality metrics framework

The software quality metrics methodology offers organizations a five step approach in support of establishing the quality requirements and identifying, implementing, analysing, and validating the process and product quality metrics for a software system.

The five steps and the process that was followed for the implementation of ACA are as follows [5].

1) Establish Software Quality Requirement.
   - Identify a list of possible quality requirements.
   - Determine the list of quality requirements.
   - Quantify each quality factor.

In the process of identifying the quality requirements we have noted that the following aspects of Software are the most significant that needs to be evaluated.

- Object oriented adherence.
- Software Complexity.
- Software/Code Size.
- Software Code Maintainability.

2) Identify Software Quality Metrics.
   - Apply the software quality metrics framework
   - Perform a cost-benefit analysis

Under each quality aspect that was identified, the following Software Metrics were selected to be calculated in order to quantify the Software quality.

- C&K metrics model [7] [8].
- Halstead metrics model and Cyclomatic complexity.
- Lines of Code, Comment Lines of Code, Token Metrics.
- Maintainability Index.

3) Implement the Software Quality Metrics.
   - Define the data collection procedures.
   - Prototype the measurement process.
   - Collect the data and compute the metric values

We have collected Software Source Codes from several University students and from some Industrial projects. The prototyped measurement process is illustrated in the Figure 2, below.

4) Analyse the Software Metrics Results.
   - Interpret the results
   - Identify software quality
   - Make software quality predictions
   - Ensure compliance with requirements

Comparing the calculated metrics values against the Standard Metrics values defined by the industry specialists the ACA interpret the metrics results and make Software quality predictions based on that.

5) Validate the Software Quality Metrics.
   - Apply the validation
   - Apply validity criteria
   - Validation procedure

To validate the Quality metrics, we have tested and compared the ACA’s metrics values against the actual metrics values and with the values generated by some other code analysing tools available in the industry. Sample Test Cases that were used to evaluate the ACA are shown in the Table 2, 3, 4 and 5.

2.2 System Overview

According to the five steps illustrated above, ACA was designed by including four core components namely;

- Object oriented metrics analysis.
- Complexity oriented metrics analysis.
- Size oriented metrics analysis.
- Maintainability metrics analysis.

Under each metric category we have considered multiple metric measurements as stated in Table 1.

Table 1 Metrics categories and selected metrics

Metric Category	**Metrics Utilized**
Object oriented metrics [6]	C&K metrics model [7] [8]
Complexity oriented metrics	Halstead metrics model & Cyclomatic complexity
Size oriented metrics	LOC, CLOC, Token metrics
Maintainability metrics	Maintainability index, Size oriented metrics

The core functionalities of this research are as follows.

- Design a well structured framework to analyze the source codes using some selected Metrics.
- Measure the metric values of a specified program or code fragment according to the designed metric based framework.
- Numerical representation of the metrics values that have been calculated.
- Graphically represent the results of the measurement to facilitate the user in a more interactive manner.
- Interpret the analysis results in a comprehensive manner to make sure that the user will have a broad understanding of the attributes of the specified code fragment.
- Provide some guidelines/suggestions based on the results of the measurement which enables the user to enhance the code quality and the other attributes of the code.
- Present customizable metrics standards so that the users can utilize their own/ organizational standards to evaluate their work.
- Provide metrics based quality notices which indicate the quality of the software.
- Measure the object oriented adherence of software based on the metrics.
- Analyze the project schedule and make metrics based schedule and actual time based estimation of the software been evaluated.
- Enable the project managers/supervisors to assign and track the tasks of the team members and enable individuals to manage their assigned tasks effectively.
- Display the source code of a selected project in a dedicated window to make it easy for the users to locate and repair the quality issues indicated by the Tool.

The above mentioned functionalities were selected with care with the intention of providing solutions for the problems that were identified during the literature survey which was done at an initial stage of the project.

To implement the proposed system functionalities we have utilized the following tools and technologies.

- Visual Studio 2010
- .Net Framework 4.0
- Microsoft SQL Server 2008
- Antlr, the Open Source Parser

Overall system functionalities of the ACA are illustrated in Figure 2.

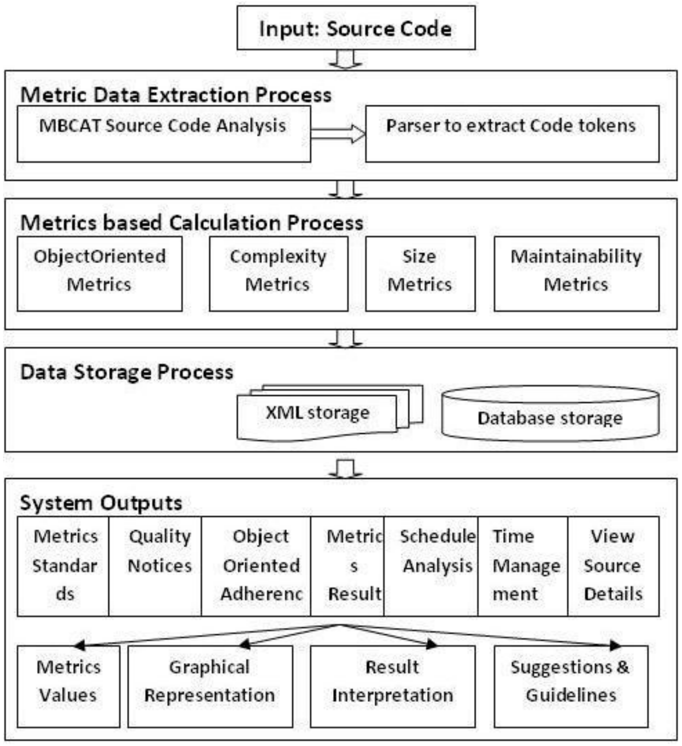


Figure 2. High level view of the ACA

2.2 User Categories and Allocated Functions

Considering the different aspects of the system we have identified four major user categories that may interact with this tool.

Executive User: This is the super user of the system who gains the full authority of the system. This user can be either from the top management of an organization who define company standards or a Project Manager who monitor and control the quality of the Project.

Developer/Programmer: This user is responsible for passing the code to calculate the various metric types. The user is able to perform all the super user functionalities except viewing cost estimations and defining standards.

Non-functional User: This user can only view the calculated metric values and a graphical representation of the metrics.

System Administrator: This user does not have any interaction with the code analysis process. He/she simply configures the user accounts and assigns specific user roles.

3. Results and Discussions

We successfully managed to implement ACA. Not only it provides a basis for project code analysis, but also performs many other functions to satisfy its users. Some of the functionalities performed by ACA are demonstrated Figure 3, Figure 4 and Figure 5.

To implement the proposed functionalities, IEEE 1061 based approach was followed so that it is evident that ACA conform the essential standards prevailing in the software engineering industry as well [5].

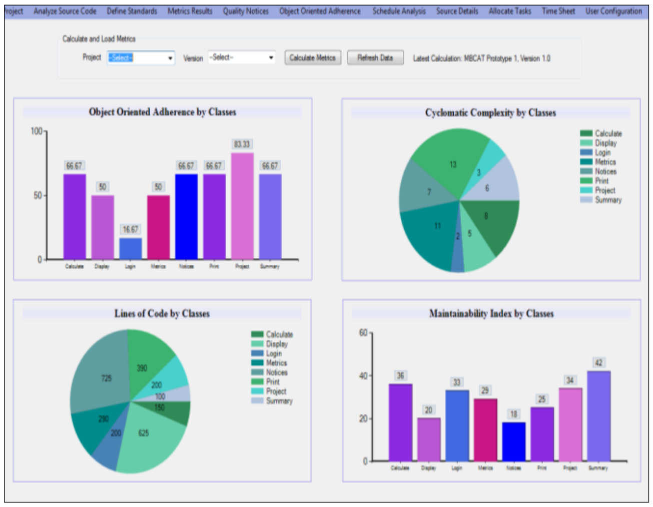
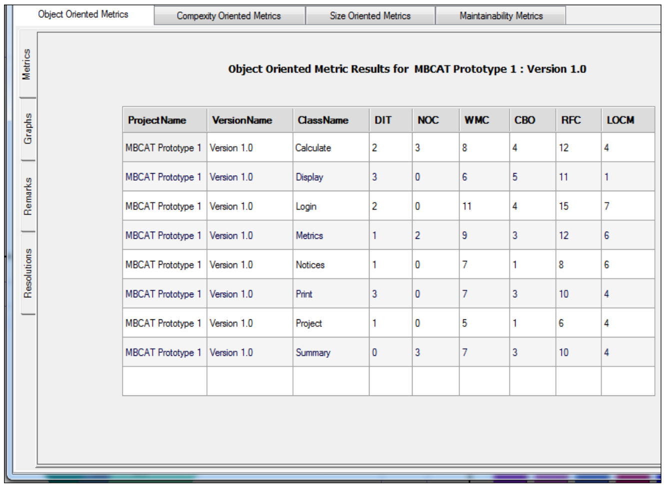


Figure 3. Main interface of ACA with sample graphs



Object Oriented Metric Results for MBCAT Prototype 1 : Version 1.0

ProjectName	VersionName	ClassName	DIT	NOC	WMC	CBO	RFC	LOCM
MBCAT Prototype 1	Version 1.0	Calculate	2	3	8	4	12	4
MBCAT Prototype 1	Version 1.0	Display	3	0	6	5	11	1
MBCAT Prototype 1	Version 1.0	Login	2	0	11	4	15	7
MBCAT Prototype 1	Version 1.0	Metrics	1	2	9	3	12	6
MBCAT Prototype 1	Version 1.0	Notices	1	0	7	1	8	6
MBCAT Prototype 1	Version 1.0	Print	3	0	7	3	10	4
MBCAT Prototype 1	Version 1.0	Project	1	0	5	1	6	4
MBCAT Prototype 1	Version 1.0	Summary	0	3	7	3	10	4

Figure 4. Metric results interface

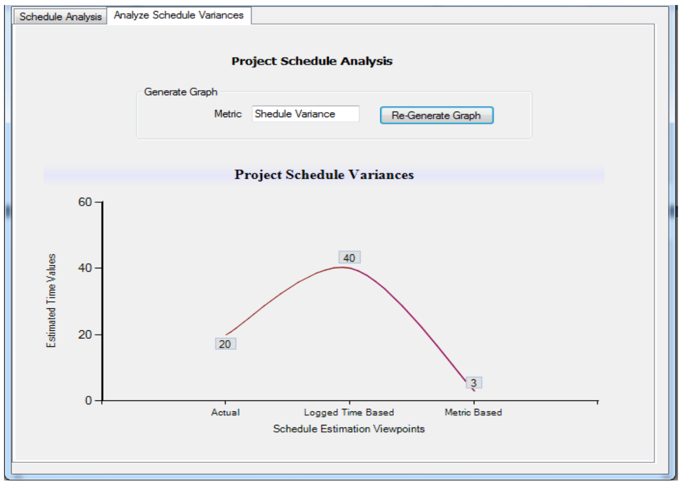


Figure 5. Schedule analysis interface

We have tested our tool against 25 source codes from a variety of applications and it was evident that the results generated by the ACA are 98% accurate. Sample test cases that we have used to evaluate the accuracy and the reliability of ACA are shown in Table 2, 3, 4 and 5.

Table 2: Metrics results comparison (i)

Size Oriented Metric	**Actual Value**	**ACA Value**	**Visual Studio Code Metrics**	**NDepend Value**
Lines of Code	200	200	210	250
Comment Lines of Code	150	150	NA	175
Unique Operators	6	6	NA	NA
Unique Operands	8	8	NA	NA
Total Operators	15	15	NA	NA
Total Operands	20	20	NA	NA

Table 3: Metrics results comparison (ii)

Object Oriented Metric	**Actual Value**	**ACA Value**	**Visual Studio Code Metrics**	**NDepend Value**
Depth of Inheritance Tree	3	3	4	NA
Number of Children	1	1	NA	NA
Weighted Methods per Class	7	7	NA	NA
Coupling Between Object Classes	1	1	NA	NA
Response for Class	8	8	NA	NA
Lack of Cohesion of Methods	6	6	NA	NA

Table 4: Metrics results comparison (iii)

Complexity Oriented Metric	**Actual Value**	**ACA Value**	**Visual Studio Code Metrics**	**NDepend Value**
Halstead Length	35	35	Not Applicable (NA)	Not Applicable (NA)
Halstead Vocabulary	14	14	NA	NA
Halstead Volume	92	92.4	NA	NA
Halstead Difficulty	6	6	NA	NA
Halstead Level	0.2	0.2	NA	NA
Halstead Effort	554	554	NA	NA
Halstead Time	31	31	NA	NA
Cyclomatic Complexity	4	4	4	2

Table 5: Metrics results comparison (iv)

Maintainability Oriented Metric	**Actual Value**	**ACA Value**	**Visual Studio Code Metrics**	**NDepend Value**
Maintainability Index	36	36	36	NA
Lines of Code	200	200	210	250
Comment Lines of Code	150	150	NA	175
Program Length	35	35	NA	NA
Program Volume	92	92.4	NA	NA
Maintenance Effort	55	554	NA	NA

Above results proves the accuracy and the reliability of the Automated Code Analyser and it is also evident that this tool supports a variety of metrics calculations which are not available in the other tools.

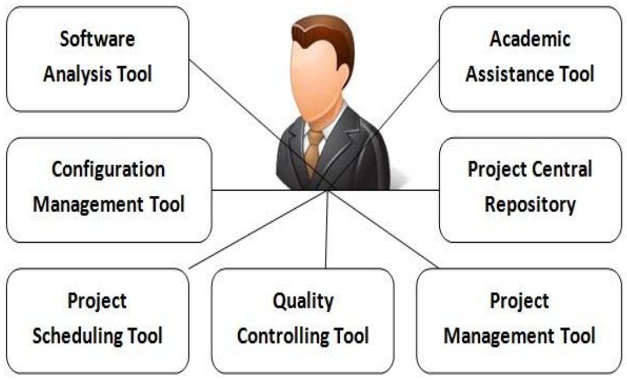


Figure 6. Main features of the system

Some of the major aspects of ACA are illustrated in the Figure 6 above, which ensures that we can acquire a huge market-share in the context of code analysing tools in software engineering industry.

4. Conclusion

In concluding the whole idea behind this research project, it can be said that what we have developed is a well structured program analysis framework based on metrics calculations, particularly for the developers and for the project managers which will enable them to perform their functions more effectively.

Our attempt was to construct a precise code analysing and estimation scheme which would have a higher explanatory power and reliability than that of current models. The idea was not just to design a simple metric calculator but to come up with an innovative solution to diminish the problems currently existing in this region of software engineering. We have successfully achieved this goal by means of the implementation of ACA.

The future releases of the ACA can be improved in the following ways.

- Modify the generated grammar files to align with the latest release of Antlr parser.
- Extend the grammar files of ACA to handle programming languages such as C, C++, Java, PHP, Ruby, Python.
- Develop a plug-in for Visual Studio for easy extraction of the code details.
- Enhance the efficiency, reliability and user friendliness of ACA.

Acknowledgments

We have taken efforts in this project. However, it would not have been succeeded without the gentle support and help of many of the individuals and organizations. We would like to extend our sincere thanks to all of them.

Similarly, all this could not be possible if not for the assistance given by the Sri Lanka Institute of Information Technology (SLIIT) management and the academics. We would like to express our special gratitude and appreciation to the all of them for all the support and guidance they have offered us throughout the Project.

References

[1] DeMacro, Tom, *Controlling software projects – management, measurement & estimation* (Yourdon Press, New York, NY, USA)

[2] DeMacro, Tom, *Software metrics: a rigorous and practical approach.*

[3] Ming Li and Carol S. Smidts. A ranking of software engineering measures based on expert opinion. *IEEE Transactions on Software Engineering, September 2003, pp. 29(9):811– 824.*

[4] D. Alderete Ash, L. Oman J. Yao, and B. P.W. Lowtber, Using software maintainability models to track code health. *In Proceedings International Conference on Software Maintenance, September 1994, pp. 154–160.*

[5] Software Engineering Standards Committee of the IEEE Computer Society, IEEE Standard for a Software Quality Metrics Methodology, *Revision of IEEE Std 1061-1992, December 1998, pp. 3-10.*

[6] Mark Lorenze, Jeff Kidd. *Obejct-oriented software metrics.* (Englewood: Prentice Hall, 1994)

[7] Chidamber and Kemerer, A Metric Suite for Object-Oriented Design, *Working Paper #249, MIT Center for Information Systems, Cambridge, MA, 1993, pp. 30.*

[8] Chidamber and Kemerer, A Metric Suite for Object-Oriented Design, *Working Paper #249, MIT Center for Information Systems, Cambridge, MA, 1993, pp. 40.*