Empirical Evaluation of Software Component Metrics

[1]AMIT KUMAR, [2]DEEPAK CHAUDHARY, [3]AVADHESH KUMAR
[1]IET, Alwar, Rajasthan, INDIA, [2]IET, Alwar, Rajasthan, INDIA, [3]Galgotias University, Greater Noida, U.P., INDIA
[1]amitk72@gmail.com, [2]deepak.se17@gmail.com, [3]kumar.avadh@gmail.com

Abstract— Many information-based legacy systems contain similar or even identical things, which are developed from scratch again and again. From the scratch, development is more expensive and can take a long time to complete. Critical applications with strict time limits may loose the market due to the delay in the development process. This has led to the evolution of a new approach, called component-based development (CBD), which uses the concept of reusability in the application development. Component-based development is the process of assembling existing software components in an application such that they satisfy a predefined functionality. Reduced development time, effort and cost are few merits of CBD.

Component based development mainly involves the reuse of already developed components. The selection of best quality component is of prime concern for developing an overall quality product. The present paper presents an empirical evaluation of some software component metrics used for reusability.

Index Terms— Component, Component Based Software Engineering, Metric, Reusability, Customizability, Portability, Interface Complexity, Understandability, Integrity, CBSD Reusability Tool.

1 Introduction

Reusability is becoming most important criteria for selecting a component for component-based systems. A highly reusable component will help in better understanding and low maintenance efforts for the application. Therefore, it is necessary to estimate the reusability of the component before integrating it into the system.

This paper aims to estimate quality characteristics of black-box components and component-based systems. The work proposes and validates metrics for reusability of the system. These estimates will help application developers to select the best quality component among others, which will eventually lead to the development of good quality product.

2 Component Based Software Engineering

CBSE embodies an element of “the buy, don’t build philosophy” that shifts the emphasis from programming software to composing software system [10]. It is also an approach for developing software that relies on software reuse and it emerged from the failure of object-oriented development to support effective reuse. Developing of software systems from the existing components have many advantages which are as follows:

1. Reliability is increased.
2. Low maintenance costs.
3. Development cost is reduced.
4. Less time to market.

A component is a language neutral, independently implemented package of software services, delivered in an encapsulated and replaceable container, accessed via one or more published interface. While a component may have the ability to modify a database, it should not be expected to maintain state information. A component is not platform-constrained nor is it application-bound [21]. A software component is a unit of packaging, distribution or delivery that provides services within a data integrity or encapsulation boundary. [3] A software component is a coherent package of software implementation that can be independently developed and delivered. It has explicit and well-specified interfaces for the services it provides and for the services it expects from the others. Also, it can be composed with other components, customizing some of their properties, without modifying the components themselves [25]. An extra effort must be paid for additional functionality of component beyond the current application’s need, to make the component more useful [16].

3 Software Component Metric

As the number of components available on the market increases, it is becoming more important to devise software metrics to quantify the various characteristics of components and their usage. Software metrics are intended to measure the software quality and performance characteristics quantitatively, encountered during the planning and execution of software development. These can serve as measures of software products for the purpose of comparison, cost estimation, fault prediction and forecasting. Metrics can also be used in guiding decisions throughout the life cycle, determining whether software quality improvement initiatives are financially worthwhile. A lot of research has been conducted on software metrics and their applications. Most of the metrics proposed in literature are based on the source code of the application. However, these metrics cannot be applied on components and component-based systems as the source code of the components is not available to application developers. Therefore, a different set of metrics is required to measure various aspects for component-based systems and their quality issues.

The authors [20] propose a set of metrics for measuring various aspects of software components like complexity, customizability and reusability. Boxall and Araban [11], considered that understandability of the component affects the level of reuse. Washizaki et al. [15], discussed the importance of reusability of components in order to realize the reuse of components effectively and proposed a Component Reusability Model for black-box components from the viewpoint of component users or application developers. Authors [15] also proposed several metrics related to these factors, namely Existence of Meta-Information (EMI), Rate of Component's Observability (RCO), Rate of Component's Customizability (RCC), Self-completeness of Component's Return Value (SCCr) and Self-completeness of Component's Parameter (SCCp).

Gill and Grover [12] has proposed interface complexity metric, based on interface signatures, constraints on the interfaces and the packaging for different context of use. For each of these aspects, a definition is given. The main drawback, it lacks an empirical evaluation and validation of the proposed metric. Sharma et al. [6] proposed interface complexity metric for software components. Author has taken into consideration interface methods and their associated properties, argument types and return types. Gill [13], has given some guidelines for high reusability for software components. These guidelines include conducting reuse assessment, performing cost-benefit analysis for reuse, adoption of standards for software components, selecting pilot projects for deployment of reuse and finally identifying the reuse metrics.

Dumke and Schmietendorf [22], have proposed a set of reusability metrics for JavaBeans components. The metrics are taken from structured and object-oriented design point of view. The authors have considered the source code to measure the metrics; henceforth it is not applicable for black-box components. Sharma et al. [2], proposed a neural network based approach to measure the reusability of a software component. Tullio Vernazza et al. [23], extended the CK metrics [24] by proposing new metrics corresponding to each CK metric. Lisa and Delugach [19], proposed the dependency representation in forms of conceptual graphs where conceptual graphs are formal, logic based, and semantic network language that are used in domain modeling and requirement modeling. Pernilla [17], has considered several factors that contribute to the complexity of large component-based software projects.

Guo [18], has proposed theory based framework for modeling component dependencies. Stafford et al. [14], has practically demonstrated a graph based representation for the dependency relationship between two or more components. Balkishan et al. [4], has introduced a set of component-based metrics, namely, Component Dependency Metric (CDM) and Component Interaction Density Metric (CIDM), which measure the dependency and coupling aspects of the software components respectively. Sharma et al. [5], has given link-list based approach to represent the dependency relationship in Component-Based System (CBS). V. Lakshmi Narasimhan et al [1] have studied series of metrics proposed by various researchers and has thoroughly analyzed, evaluated and benchmarked using several large-scale publicly available software systems.

As the diversity of the components in the market is increasing day by day, it is becoming mandatory to devise software metrics to quantify the various quality characteristics of components. Between several quality characteristics, the reusability is one of the important quality characteristics of the components. Reusability is one of the characteristics which can measure the degree of features that are reused while developing an application. There are a number of existing metrics [24, 9] available for measuring the reusability for object-oriented systems. These metrics focus on the object structure, which reflects on each individual entity such as methods and classes, and on the external attributes that measure the interaction among entities such as coupling and inheritance. The author [12], discusses the various issues concerning component reusability and its benefits in terms of cost and time-savings.

EMI and RCO metrics indicates that high value of readability will help user to understand the behavior of a component from outside the component. Many researchers had also considered similar factors for estimating reusability. Like, author [8] considered adaptability, compose-ability and complexity of a component to describe its reusability. The author [27], considered two aspects, usability and usefulness while REBOOT (Reuse Based on Object-Oriented Techniques) proposed by [28] has taken factors like portability, flexibility, understandability and confidence to assess the reusability.

Authors [6], proposed interface complexity metric for software components. Authors [2], proposed a neural network based approach to measure the reusability of a software component. The authors have considered four factors, customizability, portability, interface complexity and understandability, which is used for estimation of reusability for components. These four factors are considered as input parameters, while reusability is output parameter in order to train the network. Training and testing are performed by different number of hidden layers and neurons to get the best results. They have concluded that the neural network is able to predict the reusability of the components.

We conclude that, complex components take much time to execute and therefore they are difficult to maintain and we have to take into account the understanding of the components along with other important factors required for estimating the reusability of software components.

4 Empirical Evaluation Of Some Software Component Metrics

To obtain the values of the software component metric, an experiment is conducted on various JavaBeans components available at various websites. JavaBeans are reusable software components for Java. They are classes that encapsulate many objects into a single object (the bean). They are serializable, have a 0-argument constructor, and allow access to properties using getter and setter methods. These JavaBeans components vary from very simple and small to complex and large. These have different number of attributes and methods. We assign different weight values to these methods based on the data type of arguments or return values, used in the method. Arguments/Return types may be of primitive data types like integer, structured data types like date, string, array list, vector and complex data types like class type, built -in and user-defined components, pointers/reference and others. Therefore, based on the complexities involved in these data types, different weight values are assigned to the methods. We classify these data types in five categories, namely, very simple, simple, medium, complex and highly complex. Similarly based on the data type, we can categories properties into very simple, simple, medium, complex, and highly complex and can assign the corresponding weight values to them.

Data types are categorized as:

1. Very simple include integer, double, Boolean, float type.
2. Simple include structured data type.
3. Medium include Class type and Object type.
4. Complex includes pointers, built-in data type.
5. And highly complex includes User-defined data types.

Method having no argument (e.g. constructor) may be considered as simplest method and we assign the weight value to these methods 0.025. All other interface methods are assigned weight values depending on the type and total number of arguments and return types.

The following table shows these weight values:

Table 4.0: Weight values for Interface Methods

Data type → No. of data types ↓	Very Simple	Simple	Medium	Complex	Highly Complex
1-3	0.05	0.10	0.15	0.20	0.25
4-6	0.10	0.20	0.30	0.40	0.50
7-9	0.15	0.30	0.45	0.60	0.75
>=10	0.20	0.40	0.60	0.80	1.00

The same table can be used for getting the weight values for properties used in the component. Now, by referring to these tables, we can measure the complexity of each interface method and property, and finally by assigning 'a' and 'b' an appropriate value, the interface complexity of the component can be calculated by using equation (4.3.1).

4.1 Customizability

Customizability is defined as the ability to modify a component as per the application requirement. Better customizability will lead to a component with better reusability. It will also help in maintaining the system in the later phases. Therefore, it can be used to measure the maintainability and reusability for CBS. It may be measured on the basis of writable properties available in the component. Writable properties in Java Bean components may be recognized by set methods (Washizaki, 2003). The following formula is used to evaluate this metric:

$$\text{Customizability} = \frac{\text{No. of Set Methods}}{\text{Total number of Properties}}$$

The following table shows the number of set methods, number of properties and Customizability of various JavaBeans components:

Table 4.1: Values of customizability of various JavaBeans components

JavaBeans	Set methods	Properties	Customizability
ExplicitButton-BeanInfo	11	20	0.55
ExplicitButton-Customizer	6	12	0.50
ExternalizableBut-	6	14	0.43

ton			
OurButton	36	79	0.46
JellyBean	7	10	0.70
Juggler	9	30	0.30
JugglerBeanInfo	3	7	0.43
Atom	4	5	0.80
Molecule	8	30	0.27
MoleculeNameEd-itor	2	6	0.33

4.2 Portability

It is the ability of a component to be transferred from one environment to another with little modification, if required. It is typically concerned with reuse of component on new platforms. The component should be easily and quickly portable to specified new environments if and when necessary, with minimized porting efforts and schedules. For better reusability, component should be highly portable, means; it should be supported by several platforms. Here, for the proposed work, portability may be defined as [30]:

$$\text{Portability} = \frac{\text{No. of platforms the component can support}}{\text{Total no. of platforms that may be required by CBSS}}$$

By using this metric, we can measure that how many platforms can be supported by the component.

The following table shows the number of platforms the component can support, number of platform required and Portability of various JavaBeans components:

TABLE 4.2: VALUES OF PORTABILITY OF VARIOUS JAVABEANS COMPONENTS

JavaBeans	Platform Supported	Platform Required	Portability
ExplicitButton-BeanInfo	2	6	0.33
ExplicitButton-Customizer	2	3	0.67
Externalizable-Button	2	2	1.0
OurButton	4	6	0.67
JellyBean	5	8	0.63
Juggler	3	4	0.75
JugglerBeanInfo	5	8	0.63
Atom	4	8	0.50
Molecule	2	5	0.40
MoleculeNameEditor	5	7	0.71

4.3 Interface Complexity

We extended the approaches described in (Rotaru et al., 2005; Gill and Grover, 2004; Boxall and Araban, 2004) while proposing a new interface complexity metric for components. Proposed metric uses the signature or the behavior of the component through its interface methods and properties, which are available even without going into the internals details of the component. For the proposed metric, we consider the events and their listeners similar to the methods. We propose that the interface complexity metrics for the component will be due to the complexities involved in its interface methods and properties described above and define Interface Complexity Metric (ICM) for Component, C as:

$ICM\ (C) = a \sum CIMi + b \sum CPj$ … (4.3.1)

Where *CIMi* is the complexity of ith interface method and *CPj* is the complexity of jth property. *'a'* and *'b'* are weight values for methods and properties respectively, as complexity of an interface method may have different weight value than the complexity of a property [29]. For simplicity we have taken a= 0.8 and b= 1.2.

The following table shows the number of methods, number of properties and Interface Complexity of various JavaBeans components using equation (4.3.1):

In MoleculeNameEditor component there are following findings which are useful for calculation of interface complexity by using above mentioned formula.
Total No. of Methods = 12
Total No. of Properties = 6
Out of total 12 methods 8 methods are very simple, 2 methods are medium and 2 are simple.
IC of Total no. of methods = 0.8 * (1.6 + 1.2 + 0.8) = 2.88
Out of total 6 properties 4 are simple, one complex, and one medium.
IC of Total no. of properties = 1.2 * (0.8 + 0.4 + 0.3) = 1.8
Total ICM = 2.88 + 1.8 = 4.68

TABLE 4.3.1: INTERFACE COMPLEXITY METRIC VALUES USING

EQUATION (4.3.1)

JavaBeans	Methods	Properties	Interface Complexity
ExplicitButton-BeanInfo	34	20	6.85
ExplicitButton-Customizer	25	12	4.98
ExternalizableBut-ton	20	14	9.84
OurButton	100	79	15.16
JellyBean	23	10	5.13
Juggler	50	30	10.4
JugglerBeanInfo	15	7	2.348
Atom	12	5	1.746
Molecule	45	30	9.68
MoleculeNameEd-itor	12	6	4.68

Components are black box in nature. The source code of these components is not available. Application may interact with these components only through their well - defined interfaces. Interface acts as a primary source for understanding, use and implementation and finally maintenance for the component. Therefore, the complexity of these interfaces plays a lead role while measuring the overall complexity of the component. Complex interfaces will lead to the high efforts for understanding and customizing the components. Therefore for better reusability, interface complexity should be as low as possible. The following formula is used [30] to evaluate this criterion:

Interface Complexity = 1 – (Number of interfaces not required/ Total number of interfaces provided) …(4.3.2)

More the unrequired interfaces more will be complexity and hence less will be reusability.

The following table shows the number of interfaces not required, total number of interfaces provided and Interface Complexity of various JavaBeans components using equation (4.3.2):

TABLE 4.3.2: INTERFACE COMPLEXITY METRIC VALUES USING EQUATION (4.3.2)

JavaBeans	Interfaces Unrequired	Interfaces Provided	Interface Complexity
ExplicitButton-BeanInfo	5	29	0.83
ExplicitButton-Customizer	6	16	0.63
Externalizable-Button	5	28	0.82
OurButton	12	32	0.63
JellyBean	9	14	0.36
Juggler	10	30	0.67
JugglerBeanInfo	5	20	0.75
Atom	7	12	0.42
Molecule	8	30	0.73
Mole-culeNameEditor	4	24	0.83

4.4 Understandability

Similarly, readability can be measured by getting the observable properties from the component. Readability will help an application developer to understand the component. If a component is understandable, it will be easier to use and maintain. Therefore readability will improve the usability, reusability and maintainability of the component. It may be measured on the basis of readable properties available in the component. Readable properties in Java Bean components may be recognized by get methods. The following formula is used to evaluate this metric:

$$\text{Readability} = \frac{\text{No. of Get Methods}}{\text{Total number of Properties}}$$

The following table shows the number of get methods, number of properties and Readability of various JavaBeans components:

TABLE 4.4: VALUES OF READABILITY OF VARIOUS JAVABEANS COMPONENTS

JavaBeans	Get methods	Properties	Readability
ExplicitButtonBean-Info	5	20	0.25
ExplicitButtonCus-tomizer	9	12	0.75
ExternalizableBut-ton	10	14	0.71
OurButton	52	79	0.66
JellyBean	4	10	0.40
Juggler	21	30	0.70
JugglerBeanInfo	3	7	0.43

Atom	2	5	0.40
Molecule	23	30	0.77
MoleculeNameEditor	3	6	0.50

4.5 Integrity

Integrity is the measure of the ability of a program to perform correctly on different sets of input (Kreitzberg 1982). In a sense integrity is a measure of how well a program has been tested. A program may lack integrity if it does not account for all data options, as in the case where a program does not account for possible division by zero.

A program with a high degree of integrity should check input values to determine whether they are within practical bounds. The same concepts may be applicable with components.

$$Integrity = \Sigma[1 - threat * (1 - security)]$$

Where, threat = probability of attack (that causes failure) and
security = probability attack is repelled

The following table shows the values of threat, values of security and Integrity of various JavaBeans components:

TABLE 4.5: VALUES OF INTEGRITY OF VARIOUS JAVABEANS COMPONENTS

JavaBeans	Threat	Security	Integrity
ExplicitButtonBeanInfo	0.4	0.6	0.84
ExplicitButtonCustomizer	0.3	0.5	0.85
ExternalizableButton	0.8	0.4	0.52
OurButton	0.6	0.4	0.64
JellyBean	0.2	0.4	0.88
Juggler	0.4	0.8	0.92
JugglerBeanInfo	0.3	0.4	0.82
Atom	0.2	0.3	0.86
Molecule	0.6	0.4	0.64
MoleculeNameEditor	0.4	0.6	0.84

5 CONCLUSION AND FUTURE WORK

Component-based software development promises to reduce development costs by enabling rapid development of highly flexible and easily maintainable software systems. Higher complexity leads to high cost of maintainability. It is difficult to customize an application which is highly complex. The paper conducts an empirical evaluation on various JavaBeans components and ensures the same. From the literature review we conclude that there is no criteria to compute various quality characteristics such as maintainability, complexity, reusability, etc for component- based systems. Many researchers have proposed theoretical metrics without evaluation and validation or consider the source code of components while proposing the metrics for the above mentioned quality characteristics. The relationship between these quality characteristics to attain the overall quality as a single variable has not been explored. The main aim of the study is to estimate the quality characteristics for components and component-based systems. In this paper we propose and validate metrics for reusability of the component-based systems and components. The estimates will help application developers to select the best quality component from number of other components, which will lead to development of good quality product.

A reusability tool for CBSD can be developed using Fuzzy logic, Neuro Fuzzy logic, and Genetic Algorithm based approach.

REFERENCES

[1] V. Lakshmi Narasimhan, P. T. Parthasarathy, and M. Das, *"Evaluation of a Suite of Metrics for Component Based Software Engineering (CBSE)"*, Issues in Informing Science and Information Technology, Volume 6, 2009.
[2] Sharma, A., Kumar, R., Grover, P. S., *"Reusability Assessment for Software Components – a Neural Network Based Approach"*, International IEEE Conference 26-28 March, 2009.
[3] Microsoft Corporation.(2009) Definition of the term component. [Online]. Available: http://www.msdn.microsoft.com/repository/OIM/resdkdefinitionofthetermcomponent.asp
[4] Gill, N. S., Balkishan, *"Dependency and Interaction Oriented Complexity Metrics of Component-Based Systems"*, ACM SIGSOFT Software Engineering Notes Vol. 33 Issue 2, pp: 1-5, 2008.
[5] Sharma, A., Kumar, R., Grover, P. S., *"Estimation of Quality for Software Components - an Empirical Approach"*, ACM SIGSOFT Software Engineering Notes, Vol. 33, Issue 5, pp: 1-10, 2008.
[6] Sharma, A., Kumar, R., Grover, P. S., *"Empirical Evaluation of Complexity for Software Components"*, International Journal of Software Engineering and Knowledge Engineering (IJSEKE), Vol. 18, Issue 5, pp: 519-530, 2008.
[7] Gill, N. S., *"Importance of Software Component Characterization for Better Software Reusability"*, ACM SIGSOFT Software Engineering Notes, Vol. 31, Issue 1, pp: 1-3, 2006.
[8] Rotaru, O. P., Dobre, M., Petrescu, M., *"Reusability Metrics for Software Components"*, Proceedings of the 3rd ACS / IEEE International Conference on Computer Systems and Applications (AICCSA-05), Cairo, Egypt, pp: 24-29, 2005.

[9] Aggarwal, K. K., Singh, Y., Kaur, A., Malhotra, R., *"Software Reuse Metrics for Object-Oriented Systems"*, Proceedings of the Third ACIS Int'l Conference on Software Engineering Research, Management and Applications, pp: 48–55, 2005.

[10] Pressman, R. S., *Software Engineering: A Practitioner's Approach*, 6th Edition, McGraw Hill Book Co., 2005.

[11] Boxall, M. A. S., Araban, S., *"Interface Metrics for Reusability Analysis of Components"*, Proceedings of. Australian Software Engineering Conference (ASWEC'2004), Melbourne, Australia, pp: 40-46, 2004.

[12] Gill, N. S., Grover, P. S., *"Few Important Considerations for Deriving Interface Complexity Metric for Component-Based Systems"*, ACM SIGSOFT Software Engineering Notes, Vol. 29 Issue 2, pp: 1-6, 2004.

[13] Gill, N. S., *"Reusability Issues in Component-based Development"*, ACM SIGSOFT Software Engineering Notes, Vol. 28, Issue 4, pp: 1-5, 2003.

[14] Stafford, J. A., Alexandar, L. W., Caporuscio, M., *The Application of Dependence Analysis to Software Architecture Descriptions*, Lecture Notes in Computer Science, Vol. 2804, pp: 52-62, 2003.

[15] Washizaki, H., Hirokazu, Y., Yoshiaki, F., *"A Metrics Suite for Measuring Reusability of Software Components"*, Proceedings of the 9th International Symposium on Software Metric, pp: 211-223, 2003.

[16] Gill, N. S., Grover, P. S., *"Component-Based Measurement: Few Useful Guidelines"*, ACM SIGSOFT Software Engineering Notes, Vol. 28, Issue 6, pp: 1-4, 2003.

[17] Pernilla, E., *"Dealing with the Complexity of CBSE – Fundamental Environmental Needs"*, Chapter 18: Industrial Experience with Dassault System Component Model, J. Estublier, J. M. Favre, R. Sanlavilla, pp: 86-92, 2002.

[18] Guo, J., *"Using Category Theory to Model Software Component Dependencies"*, Proceedings of the 9th Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS, 02), pp: 185-192, 2002.

[19] Lisa, C., Delugach, H. S., *"Dependency Analysis Using Conceptual Graphs"*, In Proceedings of the 9th International Conference on Conceptual Structures, ICCS92001, pp: 117-130, 2001.

[20] Cho, E. S., Kim, M. S., Kim, S. D., *"Component Metrics to Measure Component Quality"*, Proceedings of 8th Asia-Pacific Software Engineering Conference, Macau, pp: 419-426, 2001.

[21] Sparling, M., *"Lessons Learned Through Six Years of Component-Based Development"*, Communications of the ACM Journal, Vol. 43, Issue 10, pp. 47-53, 2000.

[22] Dumke, R., Schmietendorf, A., *"Possibilities of the Description and Evaluation of Software Components"*, Metrics News, Vol. 5, Issue 1, pp: 13-26, 2000.

[23] Vernazza, T., Granatella, G., Succi, G., Benedicenti, L., Mintchev, M., *"Defining Metrics for Software Components"*, Proceedings of. World Multi-conference on Systematics, Cybernetics and Informatics, Vol. 11, pp: 16-23, 2000.

[24] Chidamber, S. and Kemerer, C., *"A Metrics Suite for Object - oriented Design"*, *IEEE Transactions on Software Engineering*, Volume 20, pp: 476-493, 1994.

[25] D" Souza, D. F., Wills, A. C., *"Objects, Components and Frameworks with UML: The Catalysis Approach"*. Addison Wesley, Reading, MA, 1999.

[26] Judith, A. C., Audrey, E. T., *"A Management Guide to Software Maintenance in COTS-Based Systems"*, a Technical Report of Mitre, Center for Air Force C2 Systems: Bedford, MA, pp: 1-35, 1998.

[27] Mili, H., Mili, F., Mili, A., *"Reusing Software: Issues and Research Directions"*, IEEE Transaction on Software Engineering, Vol. 21, Issue 6, pp: 528-561, 1995.

[28] Sindre, G., Conradi, R., Karlsson, E. A., *"The REBOOT Approach to Software Reuse"*, Journal of Systems and Software, Vol. 30, Issue 3, pp: 201-212, 1995.

[29] Arun Sharma, "*Design and Analysis of Metrics for Component-Based Software Systems*", Ph.D. Thesis, 2009.

[30] Sonu Mittal and Pradeep Kumar Bhatia, "*Framework for Evaluating and Ranking the Reusability of COTS Components based upon Analytical Hierarchy Process*", International Journal of Innovations in Engineering and Technology (IJIET), Vol. 2 Issue 4, ISSN: 2319- 1058, pp: 352-360, August 2013.

[31] Kreitzberg, Charles B. and Ben Shneiderman, "*FORTRAN Programming: A Spiral Approach*", Harcourt Brace Jovanovich, Inc., 1982.