Middle-East Journal of Scientific Research
24 (Special Issue on Innovations in Information, Embedded and Communication Systems): 06-12, 2016
ISSN 1990-9233; © IDOSI Publications, 2016
DOI: 10.5829/idosi.mejsr.2016.24.IIECS.23132

Software Complexity Measure from Software Requirements Document and Cost Driver Factors

[1]B. Arthi, [2]A. Grace Selvarani and [1]A. Sathya

[1]Department of IT, Easwari Engineering College, Chennai, India
[2]Department of CSE(PG), Sri Ramakrishna Engineering College, Coimbatore, India

Abstract: One of major criteria for the evaluation of the software quality is the software complexity. Hence, as the complexity of the software increases the accomplishment of quality of the software becomes challenging. The best option should be to predict the software complexity before the software development reaches the coding phase. When this issue is addressed the cost and time which is imposed by recoding in software development is reduced. Hence the main objective of this paper is to evaluate the complexity of the software to be developed in the early stage and to make a realistic estimate. Many factors included in COCOMO model contributes to the complexity of the software which has not been included in the existing metrics. A complexity metric, Complexity based on Requirements and Cost Drivers (CRCD) is proposed for the estimation of the software complexity from the Software Requirements Document and the COCOMO cost factors that influence the complexity of the software. CRCD evaluates the software complexity before the actual implementation phase and thus it saves cost and manpower.

Key words: COCOMO Cost drivers · Software Complexity Measure · Software Requirements Document

INTRODUCTION

The term software development refers to the process of creating a quality software system based on requirements stated by the customers within the stipulated time. It is well known fact that the quality of the software is directly related to the complexity of the software. According to IEEE [1] the measure of the software or component design or implementation that is intricate to understand and evaluate can be defined as the software complexity. Complexity can also be defined as how difficult it is to develop, understand maintain and change software.Barry Boehm proposed COCOMO basic model for cost estimation [2, 3]. This model considers the size of the software to be developed. COCOMO model does not provide complexity measure for the software; it provides an estimate for the cost and development effort. Further COCOMO-II considers Cost Driver Attributes for precise estimation of cost compared to other estimates. Most of the cost driver attributes are derived based on the complexity of the software code.

Literature Survey: This section gives an overview and the calculation methodology of various existing software complexity measures.

McCabe's Cyclomatic Metric: The initial study on the software complexity was started by Thomas J. Mc Cabe in the year 1976. He described a graph-theoretic complexity measure and illustrated how it can be used to manage and control program complexity [4]. The cyclomatic number V(G) of a graph G metric can be defined as:

$V(G) = e - n + 2p$ where,
n = Number of vertices
e = Number of edges
p = Connected components

The difficulty with this complexity measure is that, for conditional statement the complexity of the expression is never considered. Another difficulty is that this metric does not differentiate embedded loops and single loops, both the types of loops have the same complexity.

Corresponding Author: B. Arthi, Department of IT, Easwari Engineering College, Chennai, India.

Halstead's Complexity Measure: Most of the existing complexity measures are based on the code of software being developed. This section details about few existing metrics which uses the code to evaluate the software complexity. Maurice Howard Halstead in the year 1977 introduced a suite of metrics [5]. His measure was based on the count of operator and operands [6].

Program Vocabulary, $n = n1+n2$
Program Length, $N = N1+ N2$
Volume, $V= N * \log 2\ n$
Estimated Program Length $N\hat{} = n1 \log 2\ n1 + n2 \log 2\ n2$
Potential Volume, $V^* = (2+n2^*) \log2 (2+n2^*)$
Program Level, $L = V^*/V$
Effort, $E =D^*V$
Reasonable Time, $T = E/B$ min
Difficulty $= (n1/2) * (N2/n2)$

The drawback of this method is that it is not suited for fast and easy computation since it is difficult to compute and count distinct operator and operands. Hence this metric is not suited for large programs [7].

Cognitive Complexity Measures: Cognitive theory was also one of the prominent areas which were used for evaluation of complexity measure. One of the cognitive approaches was KLCID that was developed by Klemola and Rilling in the year 2004. This measure defines identifiers as a variable defined by the programmer and it is based on the identifier density (ID).

ID = Total no. of identifiers/ LOC

KLCID is calculated with the number of unique lines of code, lines that have same type and kind of operands with same arrangements of operators would be considered equal.

KLCID =

A functional size metric was proposed by Wang [8] and Shao to measure the cognitive complexity. This measure defines the cognitive weights [9] for the Basic Control Structures (BCS). Cognitive functional size [10] of software is defined as:

CFS = (Ni + No) * Wc Where,
Ni = No. of Inputs,
No = No. of Outputs and
Wc = Total Cognitive weight of software

With Cognitive knowledge as a base Kushwaha and Misra proposed a Cognitive Information Complexity Measure. Here to estimate the software complexity two main parameters were used (i) the information count with weights (ii) the Basic Control Structure with weights.

CICM = WICS * SBCS

All of the above mentioned measures are code based or they are dependent on the code of the software.

Requirement Based Complexity (RBC): Another important research on the complexity measure was done by Ashish Sharma and Dharmender Singh Kushwaha in the year 2010. They proposed a Requirements Based Complexity measure [11] based on the generic attributes derived from SRS Document [12]. The advantage of this approach is that it calculates software complexity in early phases of software life cycle. This work mainly includes the attributes extracted from the requirements document only. It does not concentrate on the other external factors that contribute to the complexity of the software.

Complexity Based on Requirements and Cost Drivers: As stated above, the following section elaborates on the complexity measure CRCD, which estimates the complexity of the yet to be developed software from the factors derived from requirements document and the COCOMO cost factors [2].

Requirement Document Factors (RDF): The following are the factors (Figure 1) that are derived from the Software Requirements Specification document.

Basic Factor (BF): Basic Factor refers to the basic input – output information about the software. The attributes that evolve from this factor are similar to that of the attributes included in function point technique [13] [14]. Following four attributes are considered:

Input: Count of information that enters the system.
Output: Count of information that leaves the system.
External Files: Count of master files included in the software.
Interface Files: Count of interface files that are used to transmit information to another system is counted.
Hence, Basic factor is calculated as;
BF = Count of Input + Count of Output + Count of External files + Count of Interface files

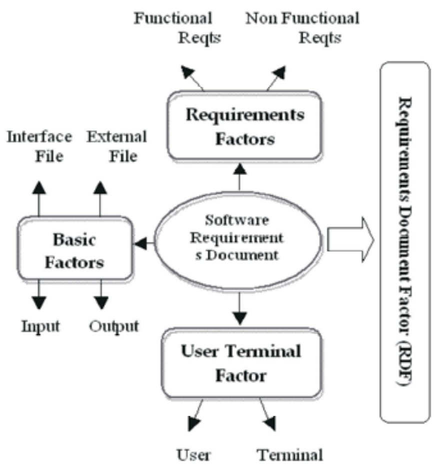

Fig. 1: Attributes of Requirement Document Factors

Requirements Factor (RF): Requirements of software can be defined as the services that the customer requires. It can also be considered as the limitations that are posed on the system during operation and development. Requirements can be broadly classified as functional and non functional requirements.

Functional Requirement: Functional Requirement states the basic functions that a software system must perform in accepting and processing the inputs and in processing and generating outputs. Functions tell what a system has to do. It is common to divide the system into number of modules or smaller units called sub functions are sub processes to ease the development process. Functional requirements are counted as:

$$FR = \sum_{i=1}^{n} \text{No. of Fn} * \text{No. of Sub Fn}$$

Non Functional Requirement: Non functional requirements define the quality related requirements for the software system. These non functional requirements can be broadly classified into three types. Each type has its own weight value. Table 1 defines the types of non functional requirements and its weight value.

Non Functional requirements are counted as:

$$NFR = \sum_{i=1}^{3} \sum_{j=1}^{n} (Type\ i * Count\ j)$$

From the above Functional Requirement (FR) and Non-Functional Requirement (NFR) count the requirements factor is calculated as follows:

RF = FR + NFR

Table 1: NFR types and weights

TYPE	WEIGHT
Mandatory	3
Desirable	2
Optional	1

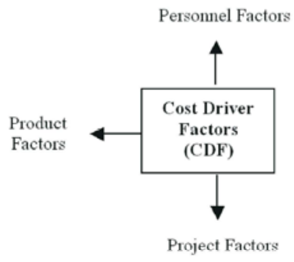

Fig. 2: Attributes of Cost Driver Factors

Users - Terminal Factor (UTF): The user- terminal factor defines the number of users accessing the system and the locations from which the users are accessing the system.

UTF= No. of User * No. of Terminal

Thus, Requirements Document Factors can be calculated by considering all the above factors. It can be mathematically defined as:

RDF = BF * RF * UTF

Cost Driver Factors: Cost drivers are features of software development that influence the complexity of the software. The cost drivers used here are taken from that of the COCOMO model. Figure 2 depicts the cost drivers used in CRCD for the calculation of the complexity of the software. Each of the cost driver attribute has ratings as Very low, Low, Nominal, High and Very High. Weights are assigned to each rating.

Product Factors (PRDF): The attributes that are considered as the product cost driver factors are:

- Required software reliability (RELY)
- Size of application database (DATA)
- Complexity of the product (CPLX)

The rating for each of the product factor attributes are described in Table 2. The value of PRDF can be calculated as the sum of rating of attributes,

$$PEDF = \sum_{i=1}^{3} CDAi$$

Table 2: Cost Driver Attributes & their values

	Rating				
Attribute	Very Low	Low	Nominal	High	Very High
RELY	0.75	0.88	1.00	1.15	1.40
DATA	-	0.94	1.00	1.08	1.16
CPLX	0.70	0.85	1.00	1.15	1.30

Table 3: Cost Driver Attributes & their values

	Rating				
Attribute	Very Low	Low	Nominal	High	Very High
ACAP	1.46	1.19	1.00	0.86	0.71
PCAP	1.42	1.17	1.00	0.86	0.70
AEXP	1.29	1.13	1.00	0.91	0.82
VEXP	1.21	1.10	1.00	0.90	-
LEXP	1.14	1.07	1.00	0.95	-

Personnel Factors (PERF): The attributes that are considered as the Personnel cost driver factors are:

- Analyst capability (ACAP)
- Programmer Capability (PCAP)
- Applications experience (AEXP)
- Virtual machine experience (VEXP)
- Programming language experience (LEXP)

The rating for each of the personnel factor attributes are described in Table 3. The value of PERF can be calculated as the sum of rating of attributes,

$$PERF = \sum_{i=1}^{3} CDAi$$

Project Factors (PJTF): The attributes that are considered as the project cost driver factors are:

- Use of software tools (TOOL)
- Multi site development (SITE)
- Required development schedule (SCED)

The rating for each of the project factor attributes are described in Table 4. The value of PRJF can be calculated as the sum of rating of attributes,

$$PRJF = \sum_{i=1}^{3} CDAi$$

The product of all the cost driver attribute values gives the estimate of the cost driver factor.

CDF = PRDF * PERF * PRJF

Table 4: Cost Driver Attributes & their values

	Rating				
Attribute	Very Low	Low	Nominal	High	Very High
TOOL	1.24	1.10	1.00	0.91	0.83
SITE	1.22	1.09	1.00	0.93	0.86
SCED	1.23	1.08	1.00	1.04	1.10

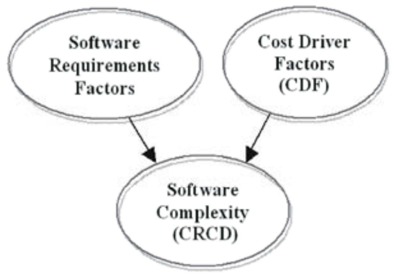
Software Requirements Factors
Cost Driver Factors (CDF)
Software Complexity (CRCD)

Fig. 3: Software Complexity

Finally, the Software Complexity CRCD can be represented as the product of Requirement Document factors and Cost driver Factors (Figure 3).

Mathematically, Software Complexity can be defined as follows,

Software Complexity (SC) = RDF * CDF

The complexity measure of the software can be estimated based on the SRS document and the cost driver factor from the COCOMO model.

RESULTS

This section validates the estimation of software complexity obtained from requirements document and cost drivers with the other existing measures. The following section describes with a sample program the comparison of the complexity estimated from the proposed methodology along with the existing measures.

Sample Program:

```
Dim Username As String
Dim Password As String
Private Sub cmdLogin_Click()
Username = "TestUser1"
Password = "TestUser1"
If Username = txtUsername
And Password = txtPassword
Then
    frmLoginSuccess.Show
    frmLoginScreen.Hide
```

```
Else
MsgBox("wrong username/password")
End If
End Sub
```

MC CABE COMPLEXITY

Number of nodes (n): 4

Number of edges (e): 4

Number of connected Component (p): 1

V (G) = e-n+2p

V (G) = 2

HALSTEAD METRICS

n1 = 8

n2 = 9

N1 = 12

N2 = 16

Program Vocabulary, n = n1+n2 = 17

Program Length, N = N1+ N2 = 28

Volume, V= N*log2n = 114.24

Difficulty = (n1/2) * (N2/n2) = 7.11

Effort, E =D*V = 812.24

KLCID MEASURE

Total no. of identifiers = 13

LOC = 12

ID = Total no. of identifiers/ LOC = 1.08

No. Of identifiers in asset of unique lines = 8

No of unique lines containing identifiers = 5

KLCID = 1.6

CRCD

Requirement Document Factors (RDF)

Basic Input Output Factor (BF)

Input Count	= 2
Output Count	= 2
Count of Interfaces files	= 0
Count. of external files	= 0

BF = 4

Requirements Factor (RF)

Functional Requirements (FR)	= 1
Non Functional Requirements (NFR)	= 0

RF = 1

RDF = 4*1 = 4

Cost Driver Factors (CDF)

Product Factors (PRDF)

CPLX	= 0.70
PRDF	= 0.70

Personnel Factors (PERF)

PCAP	= 1.42
LEXP	= 1.14
PERF	= 2.56

Project Factors (PJTF)

SCED	= 1.23
PJTF	= 1.23

CDF = 2.20

Software Complexity (SC) = RDF * CDF

= 4 *2.20

=8.80

The proposed complexity estimation model was validated with 10 sample projects. For validation purpose the SRS document of the projects were used. The following table (Table 5) provides the validation results. The Complexity values were validated against the traditional code based measures. From the values obtained it can be concluded that the CRCD measure derives the similar values as that of all the code based measures.

The following graphs gives a plot of the complexity values derived from the proposed measure versus the existing methodologies from the above table.

Table 5: Complexity values

Prog #	Mc Cabe	Halstead	KLCID	CRCD
1	1	15.5	1.2	6
2	4	28.2	1.7	11
3	3	26.2	1.4	8.8
4	2	21.7	1.1	6.2
5	8	32.6	3.2	13
6	5	27.3	1.5	10.2
7	1.8	16.5	1	4.8
8	6	23.2	2	14.5
9	9	27.8	3.9	19.8
10	4	15	1.5	11.6

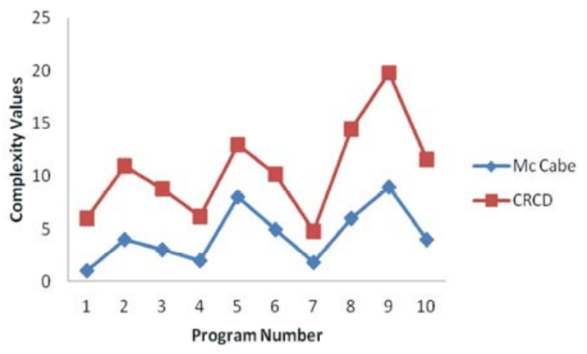

Fig. 4: Graph for CRCD versus Mc Cabes Complexity Measures

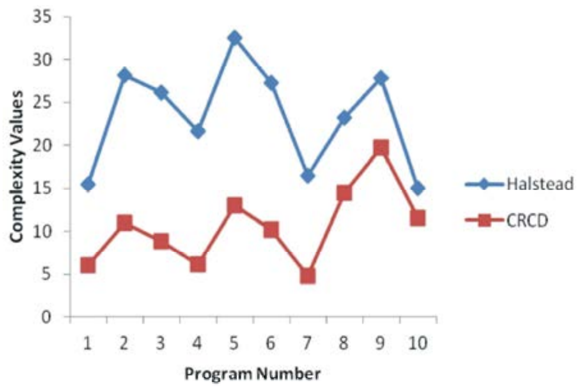

Fig. 5: Graph for CRCD versus Halstead Complexity Measures

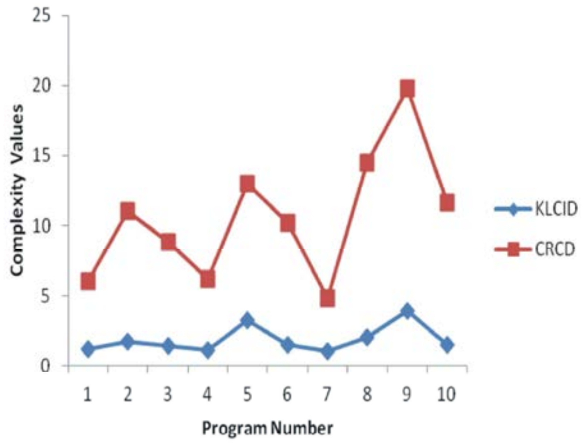

Fig. 6: Graph for CRCD versus KLCID Complexity Measures

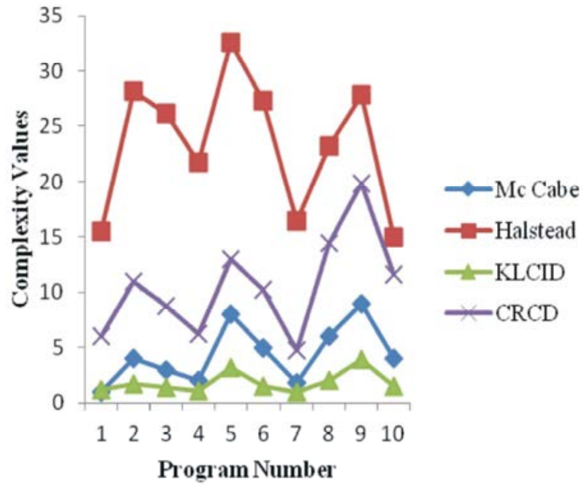

Fig. 7: Comparison Graph

The above graphs indicate that the proposed CRCD measure has the same inclination as that of the existing code based measures.

CONCLUSION

Thus this paper proposes a new estimation technique, CRCD to measure the complexity of the yet to be developed software. This method is reliable since it includes all the main attributes and factors that are required for the calculation of software complexity. Finally the results of this estimation method are compared with few of the existing method and the results are validated. The main advantages of this method are: (i) Using this metric software complexity is calculated in the early stages of the software development phase and hence it saves the development time and cost, (ii) this technique includes all the major attributes that affect or influence the complexity of the software, (iii) many attributes used in COCOMO model are included in this measure and hence this measure is comparatively more reliable than other measure.

REFERENCES

1. IEEE Computer Society: IEEE Standard Glossary of Software Engineering Terminology." IEEE std., 1990.
2. Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland, 1995. "Cost model for future software life cycle processes: COCOMO 2.0" Annuals of Software Engineering Special Volume on Software Process and Product Measurement, pp: 1.
3. Elaine J Weyuker, 1988. "Evaluating Software Complexity Measure", IEEE Transaction on Software Engineering, 14(9).
4. Sheng Yu, Shijie Zhou, 2010. "A Survey On Metric Of Software Complexity", Proceeding of 2nd IEEE International conference on Information Management and Engineering (ICIME), pp: 352-356.
5. Halstead, M.H., 1977. Elements of Software Science, Elsevier North, New York.
6. Sanjay Misra, 2007. Cognitive Program Complexity Measure, Proceedings 6th IEEE International Conference on Cognitive Informatics (ICCI'07).
7. Joseph L.F. De Kerf, 1981. APL and Halstead's theory of software metrics, In the Proceedings of the international conference on APL, 12(1).
8. Norman Fenton, 1994. Software Measurement: A Necessary Scientific Basis, IEEE Transactions on Software Engineering, 20(3).
9. Thomas J. Mc Cabe, 1976. "A Complexity Measure", IEEE Transactions on Software Engineering, SE-2(4).

10. Tom Klemola, 2000. "A Cognitive model for complexity metrics", Proceedings of 4th International workshop on Quantitative Approaches in Object Oriented Software Engineering ECOOP 00.
11. Ashish Sharma and Dharmender Singh Kushwaha, 2010. Early Estimation of Software Complexity using Requirement Engineering Document, ACM SIGSOFT Software Engineering Notes, 35(5).
12. IEEE Computer Society: IEEE Recommended Practice for Software Requirement Specifications, New York, 1998.
13. Allan J. Albrecht and John E. Gaffney, 1983. Software Function, Source Lines Of Code and Development Effort Prediction: A Software Science Validation, IEEE Transactions On Software Engineering, Se-9(6).
14. Charles R. Symons, 1988. Function Point Analysis: Difficulties and Improvements, IEEE Transactions on Software Engineering, 14(1).