Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 124 (2017) 478–486

Procedia Computer Science

www.elsevier.com/locate/procedia

4th Information Systems International Conference 2017, ISICO 2017, 6-8 November 2017, Bali, Indonesia

Cyclomatic Complexity for Determining Product Complexity Level in COCOMO II

Muhammad Asep Subandri, Riyanarto Sarno*

Department of Informatics, Instititu Teknologi Sepuluh Nopember, Surabaya, Indonesia

Abstract

Cost Construction Model (COCOMO) is an algorithmic software development cost estimation model, which requires accurate input values for each attribute. Input misjudgment on an attribute will have a great impact on the estimated effort. One of the attributes in COCOMO II is product complexity. Currently, the level of product complexity is assessed subjectively by an expert. This approach tends to be inaccurate because it is influenced by emotions, opinions and experiences. This paper proposes Cyclometer, a new approach based on the cyclomatic complexity metric to measure the product complexity level objectively. A comparison study was carried out and showed that the results comply with the expert judgments as indicated by moderate kappa statistic values.

Peer-review under responsibility of the scientific committee of the 4th Information Systems International Conference 2017.

Keywords: Software Cost Estimation; COCOMO; Product Complexity Rating; Cyclomatic Complexity

1. Introduction

According to Standish Group's *Chaos Report* [1], about 31% of investigated software projects were cancelled before completion, 53% of software project budgets exceeded original estimates by 189%, while only 16% were successfully completed on time and within budget. Among the many causes of software project failure, inaccuracies in software cost estimation have been identified as a root cause [2].

Software cost estimation is the process of predicting the amount of costs and resources required to develop software as realistically as possible. If the estimation is higher than the actual value, resources will be wasted. If the estimation

* Corresponding author. Tel.: +0-000-000-0000 ; fax: +0-000-000-0000 .
E-mail address: riyanarto@if.its.ac.id

1877-0509
Peer-review under responsibility of the scientific committee of the 4th Information Systems International Conference 2017
10.1016/j.procs.2017.12.180

is lower than the actual value, a financial loss will be incurred and the development team must work under pressure to complete the project on time. Eventually, the software will be delivered over budget and after the deadline, or the project is terminated [3].

In recent years, several researchers from the fields of software engineering and project management have attempted to find approaches or models to enhance software cost estimation accuracy [4], but none of them have achieved 100% accuracy[5]. Currently, attempts are still being made to make software development cost estimation more precise and closer to the actual value. Several models have been proposed, which can be categorized into three groups: algorithmic, non-algorithmic and artificial intelligence models [5]. The algorithmic model is the oldest and best-known method. Compared to other estimation models, algorithmic models are relatively accurate [6]. Some examples of algorithmic models are COCOMO, SLIM, Function Points and Use Case Points [3]. To obtain an accurate estimation, the algorithmic models require accurate input for each attribute, such as source lines of code, number of interfaces, software complexity, database size, programmer capability, etc. Misjudgment on one of the attributes will have a great impact on the estimated development cost [3,7]. For instance, a misjudgment on product complexity level in COCOMO II from "very low" to "extra high" means a 238% increase in the estimation of the effort.

COCOMO II uses statistical analysis of historical data and expert judgment to estimate software attributes. Currently, the product complexity level in COCOMO II is rated subjectively by an expert. This subjective method tends to provide biased, inconsistent and error-prone results that affect the estimation accuracy of COCOMO II. The purpose of the present work was to develop an objective and measurable product complexity rating. We propose a method called Cyclometer, which uses a UML activity diagram derived from the software requirements and McCabe's cyclomatic complexity metric to determine the product complexity level.

2. COCOMO II

In 1981, Boehm introduced COCOMO, a method to estimate software development cost. COCOMO was designed for sequential software development techniques. Nowadays, with the concepts of reusable and ready-to-wear software components, implementing the original COCOMO can be problematic. In 1997, Boehm developed COCOMO II in response to the latest software development techniques and paradigm, and to improve the method's estimation accuracy. In [6] it is shown that COCOMO II obtains better results compared to other algorithmic development cost estimation models. To estimate effort, COCOMO II's post-architecture model takes input from several software attributes that significantly affect the estimation of software development projects [8, 9]. The attributes are: software size (KLSOC); 5 scale factors (*precedentedness*, *development flexibility*, *architecture/risk resolution*, *team cohesion*, *process maturity*); and 17 effort multipliers, which are grouped into four categories:

- Product attributes (*required software reliability*, *database*, *product complexity*, *developed for reusability*, *documentation match to life-cycle needs*).
- Computer attributes (*execution time constraint*, *main storage constraint*, *platform volatility*).
- Personnel attributes (*analyst capability*, *programmer capability*, *personnel continuity*, *application experience*, *platform experience*, *language and tool experience*).
- Project attributes (*use of software tools*, *multisite development*, *required development schedule*)

All scale factors and effort multipliers are assigned qualitative rating levels that range from "very low" to "extra high". This paper only discusses determination of the product complexity level; for determination of other attribute levels, see the guidelines in [10]. COCOMO II divides software complexity into five major aspects: *control operations*, *computational operations*, *device-dependent operations*, *data management operations*, and *user interface management operations*. To determine the complexity of a product, select one of the aspects or a combination of aspects that characterizes the product, determine the product complexity level subjectively and then calculate the average of the selected aspect ratings.

In this work, only the *control operations* aspect was used. Table 1 shows the guidelines to determine the product complexity rating. Based on the table, product complexity will be rated "very low" if there is only one programming operator; it will be rated "low" if there are a programming operator and a nested programming operator; it will be rated "nominal" if there are a programming operator, a nested programming operator, and inter-module control; it will

be rated “high” if there are a programming operator, a nested programming operator, inter-module control, distributed processing; and so on.

Table 1. COCOMO II Product Complexity Rating Guidelines.

Rating	Programming operator	Nested programming operator	Inter-module control	Distributed processing	Reentrant and recursive	Real-time control
Very low	Yes	No	No	No	No	No
Low	Yes	Yes	No	No	No	No
Nominal	Yes	Yes	Yes	No	No	No
High	Yes	Yes	Yes	Yes	No	No
Very high	Yes	Yes	Yes	Yes	Yes	No
Extra high	Yes	Yes	Yes	Yes	Yes	Yes

3. Cyclomatic Complexity

One of the most popular software complexity metrics for measuring the complexity of a program is cyclomatic complexity (CC), developed by McCabe in 1976 [11]. The underlying theory is that the greater the number of paths through a module, the higher its complexity. Cyclomatic complexity is calculated using a control flow graph (CFG), generated from the source code. To make a CFG, create a node for every statement in the source code and assign a number to the nodes and after that connect every node to an edge (represented by arrows) based on their flow. Finally, calculate the cyclomatic complexity with equation (1). The cyclomatic complexity value must be kept below 10. This indicates well-structured and well-written code, high testability, low cost and effort to build and maintain. If the cyclomatic complexity number is above 10, the source code is complex, has low testability, and high cost and effort to build and maintain.

$$CC = E - N + 2 \tag{1}$$

Where, E = number of edges, N = number of nodes

The measurement of software complexity using cyclomatic complexity can be done not only based on the coding phase. There have been several studies using cyclomatic complexity to measure software complexity based on the design phase. Chouhan et al. [12], Boghdady et al. [13], proposed the use of activity diagrams to measure software complexity and generate software test cases using cyclomatic complexity. Nigam et al. [14] also proposed the same approach but based it on a storyboard. Zapata et al. [15] proposed the use of flow graphs to measure the complexity of a structured system using cyclomatic complexity.

4. Methodology

Most of the software complexity measurements in previous works were conducted to assess software quality. In this work, the measurement of software complexity was performed to determine the product complexity level in COCOMO II. The proposed software complexity measurement method is called Cyclometer and is based on activity diagrams derived from the software requirements. The advantage of Cyclometer is that it does not require previous similar project data or expert judgment, and provides a measurable rating, not a subjective rating.

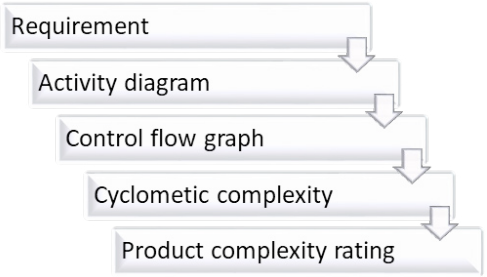

Fig. 1. Proposed cyclometer process.

Fig. 1 shows the process of the Cyclometer approach, which consists of 5 steps. Firstly, collect the software project requirements. Secondly, design an activity diagram based on the requirements. Thirdly, transform the activity diagram into a control flow graph using the following rules: (1) transform the initial, activity, decision, and final node notation in the activity diagram to graph nodes; (2) transform the control flow in the activity diagram to graph edges; (3) if there is a fork node or join node in the activity diagram, connect the node in the control flow graph directly with an edge without considering the fork node or join node. Fourthly, calculate the cyclomatic complexity of the control flow graph that was built in the previous step. The last step is to determine the product complexity rating in COCOMO II based on the obtained value of cyclomatic complexity. Translation of the cyclomatic complexity value to the product complexity rating can be seen in Table 2 [16, 17].

Table 2. CC Translation to COCOMO II product complexity rating.

CC	CPLX rating
1 to 4	Very low
5 to 10	Low
11 to 20	Nominal
21 to 40	High
41 to 50	Very high
> 50	Extra high

As case study, an online shopping software project was selected. Table 3 below shows the software requirements. Fig. 2 (a) shows the transformation of the software requirements into an activity diagram, Fig. 2 (b) shows the conversion of the activity diagram to a control flow graph.

Table 3. Online shopping system requirements.

ID	Requirements
RQ01	The system has to provide browsing options to see product details.
RQ02	The system has to provide a search facility to find products.
RQ03	The system has to display all matching products based on the search.
RQ04	The system has to display detailed information about the selected products.
RQ05	The system has to provide a shopping cart during online purchase.
RQ06	The system has to allow the user to add/remove products in the shopping cart.
RQ07	The system has to allow the user to confirm the purchase.
RQ08	The system has to enable the user to enter payment information.
RQ09	The system has to display a detailed invoice for the current order once it is confirmed.

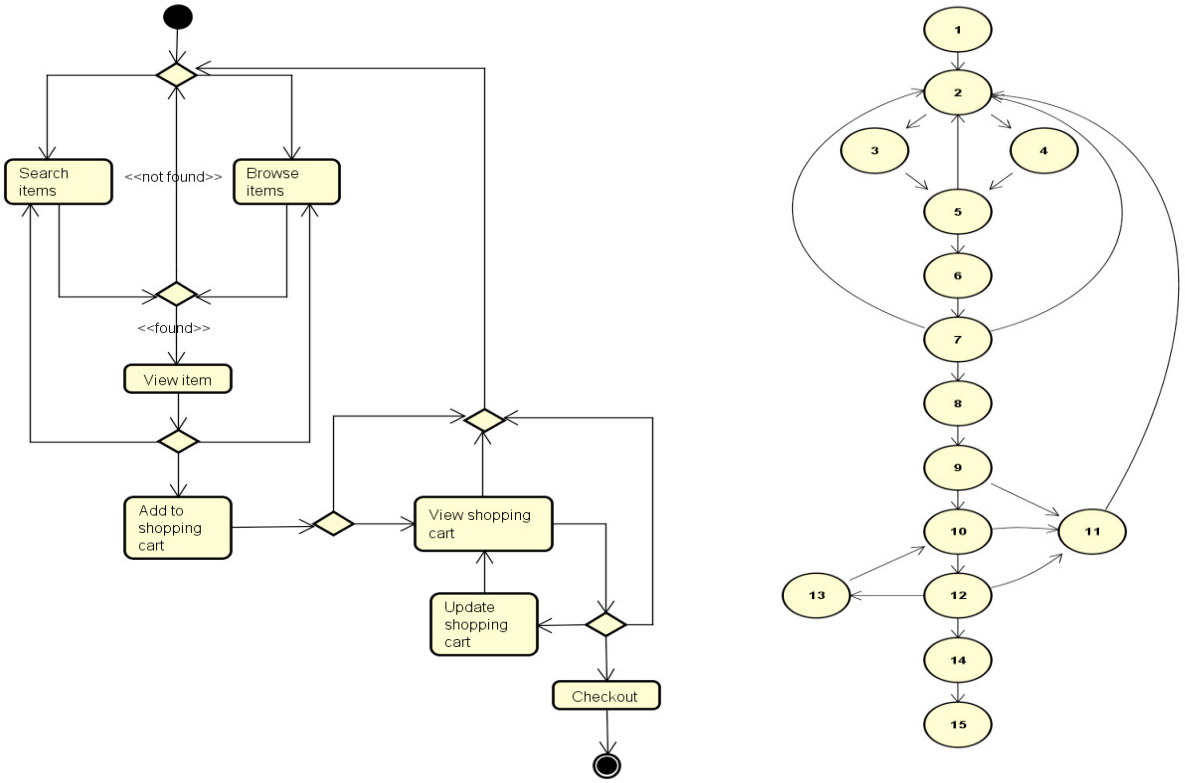

Fig. 2. (a) Activity diagram of online shopping project; (b) control flow graph of online shopping project.

Based on the above control flow graph, the cyclomatic complexity number of the online shopping software project can be calculated. Fig. 2 (b) shows that there are 22 edges and 15 nodes. Using equation (1), the cyclomatic complexity value obtained is 9. This cyclomatic complexity value translated to product complexity in COCOMO II is "low" (see Table 2).

$$CC = 22 - 15 + 2 = 9$$

5. Results

This section compares the determination of the cyclometer product complexity level with subjective judgment by experts. After obtaining the product complexity level using cyclometer, judgments from three experts about the product's complexity were collected. The experts that were asked for a judgment were experienced programmers with the capability to build complex software. They were asked to provide product complexity ratings according to the COCOMO II guidelines (Table 1). Because this is a subjective process, the judgments given by the three experts differed from each other. To select which was the best judgment alternative from the three experts, multiple criteria decision analysis (MCDA) with analytic hierarchy process (AHP) as criteria weighting technique [18] was used. MCDA is a decision-making process approach used to determine the optimal choice between conflicting goals. MCDA is commonly used in everyday life, for example trying to decide where to study considering housing costs, university ranking, student life, and study facilities [19]. After the best judgment alternatives were found, we compared the results to Cyclometer using kappa statistic [20]. Kappa statistic can be calculated with equation (2):

$$\kappa = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)} \tag{2}$$

Where, *Pr(a)* = observed agreement, *Pr(e)* = expected agreement

The kappa statistic result can be interpreted as follows: values between 0 to 0.20 indicate no agreement among two observers, 0.21 to 0.39 indicate minimal agreement, 0.40 to 0.59 indicate weak agreement, 0.60 to 0.79 indicate moderate agreement, 0.80 to 0.90 indicate strong agreement, and values between 0.91 to 1.00 indicate almost perfect agreement [20].

Table 4. Product complexity rating results of 20 software projects using expert judgments.

Project	Expert 1	Expert 2	Expert 3
1	VL	VL	VL
2	VL	VL	VL
3	VL	VL	VL
4	VL	VL	VL
5	L	L	L
6	L	L	L
7	VL	VL	VL
8	VL	L	L
9	VL	L	VL
10	L	L	L
11	VL	VL	VL
12	VL	VL	VL
13	VL	VL	VL
14	L	L	L
15	VH	VH	EH
16	N	N	N
17	H	H	N
18	N	L	N
19	H	VH	H
20	VH	H	H

VL = very low, L = low, N = nominal, H = high, VH = Very high, EH = extra high.

Table 4 shows 20 software project product complexity ratings from the three experts. There were 7 software projects that received different judgments. Table 5 illustrates the use of MCDA to select the best judgment alternative among those of the three experts. The first step of the MCDA approach is to determine the criteria. In this case, criteria derived from the product complexity rating guidelines in COCOMO II were used. Then, each complexity criterion was given a weight. The three experts were asked to score every criterion when they were asked to provide their judgments. The criterion score for each expert was then multiplied by the weight, followed by summing up all scores. The overall score with the highest value is the best alternative. From the example, the best alternatives are the judgments given by Expert 1. This MCDA approach was also done for the 6 other software projects with different judgments. The full results can be seen in Table 6, along with the product complexity ratings using Cyclometer.

Table 5. Example of selecting best expert judgment alternative using MCDA.

	Criteria	Scores from Expert 1	Scores from Expert 2	Scores from Expert 3	Weights	Expert 1	Expert 2	Expert 3
1	Programming operator	80	50	60	0.43	34.40	21.50	25.80
2	Nested programming operator	40	70	60	0.64	25.60	44.80	38.40

	Criteria	Scores from Expert 1	Scores from Expert 2	Scores from Expert 3	Weights	Expert 1	Expert 2	Expert 3
3	Inter-module control	35	30	25	1.01	35.35	30.30	25.25
4	Distributed processing	15	10	10	1.60	24.00	16.00	16.00
5	Reentrant and recursive	5	5	3	2.50	12.50	12.50	7.50
6	Real-time control	0	0	0	3.82	0.00	0.00	0.00
Overall score of the alternatives						131.85	125.10	112.95

Table 6. Product complexity rating comparison between cyclometer and best alternative judgments.

Project	Cyclometer	Best judgment
1	VL	VL
2	VL	VL
3	VL	VL
4	VL	VL
5	L	L
6	L	L
7	VL	VL
8	L	L
9	VL	VL
10	L	L
11	VL	VL
12	VL	VL
13	VL	VL
14	L	L
15	EH	VH
16	N	N
17	H	H
18	N	L
19	EH	VH
20	H	H

Table 7. COCOMO II Complexity Rating Agreement Between Cyclometer and Best Alternative Judgments

		Best judgments						
		1	**2**	**3**	**4**	**5**	**6**	
Cyclometer	**1**	9	0	0	0	0	0	9
	2	0	5	0	0	0	0	5
	3	0	1	1	0	0	0	2
	4	0	0	0	2	0	0	2
	5	0	0	0	0	0	0	0
	6	0	0	0	0	2	0	2
		9	6	1	2	2	0	20

After obtaining a uniform judgment value, as shown in Table 6, kappa statistics was used to measure Cyclometer's reliability. According to the kappa statistic data, as shown in Table 7, the complexity of 17 (85%) software projects has agreement between the cyclometer value and the expert judgments, while the complexity of 3 (15%) software projects has disagreement between the cyclometer value and the expert judgments. Using equation (2) to calculate the kappa value, the result was 0.78, which can be interpreted as moderate agreement between Cyclometer's rating and the expert judgments.

6. Conclusion

In this paper, Cyclometer was proposed, an approach to get an objective and measurable product complexity rating in COCOMO II. To evaluate the reliability of this approach, its results were compared with the subjective judgment of experts. Three experts were asked to rate the product complexity of 20 software projects. As a result, the three experts gave different judgments on 7 projects. To find the best judgment alternatives, the MCDA approach was used. After getting uniform values, the results of Cyclometer were compared with the expert judgments using kappa statistic. The agreement between the Cyclometer and the expert ratings had a kappa value of 0.78, which can be interpreted as moderate. This means that Cyclometer can be used as an alternative approach to determine product complexity in COCOMO II. In a future work, we plan to use the basic flow of requirements in SRS documents to determine COCOMO II product complexity ratings without converting them into activity diagrams and control flow graphs.

Acknowledgements

The authors would like to thank the Indonesia Endowment Fund for Education (LPDP) for their support and financial assistance.

References

[1] The Standish Group. The Standish group: the chaos report. *Proj Smart*. 2014:16. doi:10.1016/S0895-7061(01)01532-1.

[2] Attarzadeh I, Ow SH. Improving estimation accuracy of the COCOMO II using an adaptive fuzzy logic model. *Fuzzy Syst (FUZZ), 2011 IEEE Int Conf*. 2011:2458-2464. doi:10.1109/FUZZY.2011.6007471.

[3] Gharehchopogh FS, Maleki I, Talebi A. Using hybrid model of Artificial Bee Colony and Genetic Algorithms in Software Cost Estimation. *2015 9th Int Conf Appl Inf Commun Technol*. 2015:102-106. doi:10.1109/ICAICT.2015.7338526.

[4] Sarno R, Sidabutar J, Sarwosri. Comparison of different Neural Network architectures for software cost estimation. *2015 Int Conf Comput Control Informatics its Appl*. 2015:68-73. doi:10.1109/IC3INA.2015.7377748.

[5] Gharehchopogh FS, Rezaii R, Arasteh B. A new approach by using Tabu search and genetic algorithms in Software Cost estimation. *2015 9th Int Conf Appl Inf Commun Technol*. 2015:113-117. doi:10.1109/ICAICT.2015.7338528.

[6] Toka D, Turetken O. Accuracy of contemporary parametric software estimation models: A comparative analysis. *Proc - 39th Euromicro Conf Ser Softw Eng Adv Appl SEAA 2013*. 2013:313-316. doi:10.1109/SEAA.2013.49.

[7] Singh BK, Tiwari S, Mishra KK, Misra AK. Tuning of Cost Drivers by Significance Occurrences and Their Calibration with Novel Software Effort Estimation Method. *Adv Softw Eng*. 2013;2013(1):1-10. doi:10.1155/2013/351913.

[8] Ramacharan S, Rao KVG. Software Effort Estimation of GSD Projects Using Calibrated Parametric Estimation Models. *Proc Second Int Conf Inf Commun Technol Compet Strateg - ICTCS '16*. 2016:1-8. doi:10.1145/2905055.2905177.

[9] Sarno, R., Sidabutar, J. S. Improving the Accuracy of COCOMO's Effort Estimation Based on Neural Networks and Fuzzy Logic Model. *Int Conf Information, Commun Technol Syst*. 2015:197-202. doi:10.1109/ICTS.2015.7379898.

[10] Boehm BW, Clark, Horowitz, et al. *Software Cost Estimation with Cocomo II*. New Jersey, USA: Prentice Hall; 2000.

[11] Tiwari U, Kumar S. Cyclomatic complexity metric for component based software. *ACM SIGSOFT Softw Eng Notes*. 2014;39(1):1-6. doi:10.1145/2557833.2557853.

[12] Chouhan C, Shrivastava V, S Sodhi P. Test Case Generation based on Activity Diagram for Mobile Application. *Int J Comput Appl*. 2012;57(23):4-9. doi:10.5120/9436-3563.

[13] Boghdady, Pakinam N., Badr, Nagwa L., Hashem, Mohamed, and Tolba MFT. A Proposed Test Case Generation Technique Based on Activity Diagrams. *Int J Eng Technol*. 2011;11(3):37-57.

[14] Nigam A, Nigam B, Kumar Vatsa D. Generating all Navigational Test Cases using Cyclomatic Complexity from Design Documents for

Mobile Application. *Int J Comput Appl*. 2012;40(12):40-44. doi:10.5120/5020-7354.

[15] Zapata F, Akundi A, Pineda R, Smith E. Basis path analysis for testing complex system of systems. *Procedia Comput Sci*. 2013;20:256-261. doi:10.1016/j.procs.2013.09.270.

[16] Laird LM, Brennan MC, IEEE Computer Society., John Wiley & Sons. *Software Measurement and Estimation : A Practical Approach*. New Jersey, USA: Wiley-Interscience; 2006.

[17] J. MALHOTRA J, S. TIPLE B. *Software Testing and Quality Assurance*. Pune, India: Nirali Prakashan; 2008.

[18] Thokala P, Devlin N, Marsh K, et al. Multiple criteria decision analysis for health care decision making - An introduction: Report 1 of the ISPOR MCDA Emerging Good Practices Task Force. *Value Heal*. 2016;19(1):1-13. doi:10.1016/j.jval.2015.12.003.

[19] Aldrin Wiguna K, Sarno R, Ariyani NF. Optimization Solar Farm site selection using Multi-Criteria Decision Making Fuzzy AHP and PROMETHEE: case study in Bali. In: *2016 International Conference on Information & Communication Technology and Systems (ICTS)*. IEEE; 2016:237-243. doi:10.1109/ICTS.2016.7910305.

[20] McHugh ML. Interrater reliability: the kappa statistic. *Biochem Medica*. 2012:276-282. doi:10.11613/BM.2012.031.