Complexity Assessment based on UML-Activity Diagram

Maushumi Lahon, Uzzal Sharma

Abstract: Assessing complexity can significantly contribute to the attainment of the various quality attributes associated with a system. The avoidable complexity can be identified and reduced on the basis of the assessment. It holds the key to success of the system being developed. Various evaluation methods exist which have specific objectives and basis and all contribute to enhance product quality. In this paper a Complexity Assessment approach based on Activity Diagrams (CAAD) is proposed to evaluate the process view of the architecture of a system. The proposed approach estimates the complexity of the system/class/function from the UML representation of the process view of the architecture in the form of activity diagrams. This complexity measure may be used to assess and estimate the time and effort required to develop the system. This approach can estimate the coding complexity in terms of size without actually developing the code for the system/class/function. The paper is on calculating a complexity factor C from the given activity diagram and further develop a relationship between C and LOC metrics.

Index Terms: Software development, Activity Diagrams, Activity flow graph, Complexity, Metrics

I. INTRODUCTION

Complexity is an area related to every domain of work. In the area of software development, complexity has great impact on the development effort and cost. So to discuss the challenges of this area, research works are being carried out at different levels with different measures. It becomes difficult to achieve the desired quality of software as they are growing in size and thus increasing the complexity of developing them [1].

- This work presents an approach to evaluate the architecture of a software system from the process view architectural representation of the system. The approach is based on 4 + 1 view representation of architecture proposed by Kruchten in his work [2].
- The objective of the approach is to evaluate the architecture by calculating the complexity of the system from the process view representation of the architecture by using metrics to calculate complexity. Unified Modelling Language (UML) is used, as it is considered as industry standard **modelling language with a rich graphical notation, and comprehensive set of** diagrams **and elements.** Both the structural and behavioural aspects of the system can be represented using the various UML diagrams.
- The proposed approach "Complexity Assessment based on Activity Diagrams" (CAAD) evaluates the system/class/function to determine the development complexity. This could be used as an indicator for estimating the effort in implementing the other quality attributes like modifiability, cost and benefit, maintainability used in other approaches for evaluating the architecture.
- The focus of this work is to find a measure of complexity of the system/class/function in terms of developing it. This measure can facilitate the process of estimating the effort and time required to develop the system without actually developing the code. It is an essential part of system development to estimate the size of the system which introduces different types of complexity in developing the system. This complexity could be in terms of effort and time which has to be converted to person-month estimate. This finally can contribute to the cost estimation of developing the system.
- The paper proposes the estimation of a complexity measure C from the activity diagram representation. This activity diagram could be of a system, a class or a function. The activity diagram is converted to an activity flow graph from which the metrics for calculating C is derived. Higher C value indicates more development complexity.
- Finally, an implementation of the approach on a project was made to develop a relationship between the C value and LOC metrics to establish that higher C values do indicate more complexity.

The rest of the sections are organised as follows: Section II covers the review of the previous works on metrics and complexity, Section III gives the methodology of calculating the complexity value from the activity diagrams along with a case study. Section IV highlights the various points with respect to the proposed complexity measure approach in comparison with complexity measures derived from other UML diagrams, section V presents the implementation of the proposed approach on a project to calculate the complexity value from different activity diagrams used in the selected project. Section VI presents the Analysis and Results from the implementation and finally section VII draws the conclusion and suggests directions for future work.

Revised Manuscript Received on 30 July 2019.

* Correspondence Author
Maushumi Lahon*, Depatrment of CSE, Assam Engineering Institute, Guwahati,Assam.
Uzzal Sharma, School of Technology,Assam Don Bosco University, Guwahati,Assam.

II. PREVIOUS WORK

The term complexity is a familiar term and forms a part of almost every domain. More functionality induces more complexity, which in turn is related to effort and cost. Moreover, complexity is directly related to quality aspect, more requirements in quality factors introduce more complexity in the development phase. Researchers working in this area of complexity in different domains have developed metrics and measures to assess and deal with the context. In the software engineering domain also design complexity and code complexity is a significant area of concern and is a matter of interest even today. Weyukars' work on evaluating software complexity measures suggest that complexity measures was an area of interest during that period and software complexity measures already existed then which were evaluated in the work [3]. Zuse in his book has described and analysed different software complexity measures and methods and states that complexity is a multidimensional property which cannot be estimated by a single magic number [4]. Review of complexity metrics [5], use of complexity metrics in for analysis in different areas [6] and design of different complexity metrics [7,8,9] clearly establish the fact that, study in this area is of much interest to the research community. A work on assessing architecture complexity way back in 1998, proposed the Interactive Architecture Pattern Recognition (IAPR) system for locating user-specified at varying levels of abstraction patterns within software architecture. No concept of metrics was involved in this [10]. Software Architecture Complexity Model (SACM) was proposed based on theories from cognitive science and system attributes that have proven to be indicators of maintainability in practice [11]. The model emphasise difficulties in understanding software architecture and the elements that complicate the verification of implemented and intended architecture. Tong Yi et.al in their work on comparisons of metrics for UML class diagrams represented a comparison based on different aspects [12]. The author considers six existing metrics to measure the complexity for UML class diagrams and compares the metrics on the basis of different viewpoints. Kang et.al uses weighted class dependence graphs to represent a class diagram. The structural complexity of the class diagram is calculated based on entropy distance which is an objective measure [13, 14]. Marchesi’s proposed metrics focused on use-case diagrams and class diagrams which intends to measure the orientation of the object and quality of the system. The proposed metrics aims to measure complexity, responsibility balancing among packages and classes, as well as cohesion and coupling among the entities of the system [15]. Sengupta and Kanjilal proposed the Component Architecture Complexity Measurement Metrics (CACMM) on UML component diagrams. They used Component Architecture Graph for graphically representing the component diagram. This graphical representation was analysed to measure complexity at individual component level, component to component level and the overall architecture. The metrics can be used to redesign to minimise coupling and create highly cohesive components [16]. Complexity of a system can be assessed at two different levels, the architecture level and the behaviour level [17]. At the architecture level both formal and informal notations are used for assessment, but most often complexity is assessed at the code level applying the industry standard code metrics as McCabe Complexity metrics [18], Halstead Complexity Metrics [19] and Zage Complexity Metrics [20]. Other approaches to assess complexity are based on calculating the program elements like data types, number of methods and arguments of a function. Mesa et.al [21] in their work on complexity assessment measure has based their assessment measure on the tasks required for assembling and disassembling. The objective of the work is to measure complexity so that it can be reduced to give significant impact on the lifecycle of the system. The importance of complexity measure in the early stages have been seen in other works stating that complexity measure is an early indicator of the effort required for testing, implementing quality factors and minimising cost [22,23,24]. The presence of subjective component in complexity analysis is also considered to be a significant aspect which requires to be addressed while measuring system complexity [25]. The proposed approach considers the activity diagram from UML diagrams to capture the process view of the architecture. Literature reveals that there are different variations and representations of the UML notations depending on the purpose and research objective. As basic UML diagrams do not represent the artifact necessary for different types of evaluation, the notations are tailored according to the requirements of research. Such representation is seen in dependability evaluation of an architecture using Dependability Modelling and Analysis (DAM) annotations [26]. Similar work on reliability evaluation is found using DAM profile [27]. In connection to activity diagrams Petri net representation of UML 2 activity diagrams and Coloured Petri Net (CPN) representations are found in the literature review. CPN representation was used for execution validation [28]. The proposed complexity assessment approach CAAD is based on UML activity diagram to represent the system complexity from the dynamic aspect using a simple node and edge concept without going into the complex representations like Petri nets or CPN.

III. COMPARISON OF THE PROPOSED APPROACH WITH OTHER COMPLEXITY MEASUREMENT METHODS USING UML DIAGRAMS

There are different methods proposed to measure complexity of a system using UML diagrams. As no relevant work on measuring system/class/function complexity based on activity diagram was found in the review, a comparison of the proposed approach is made with complexity assessment approaches based on use-case diagrams and class diagrams proposed by Marchesi [15].

In case of complexity assessment based on Use-case diagram, four metrics were proposed –

- Number of Use Cases
- The overall number of communications among use cases and actors
- The total number of communications among use cases and actors, without redundancies introduced by extend and use relationships.
- Estimate of the global complexity of the system (assumption is that the key complexity measurement is the number of independent communications, metric

UC3).

From the proposed metrics it is clear that the estimate of complexity is based on what is to be done by the system, not how. Complexity is introduced when the how component is decided as the resources required can be estimated. This estimate covers the different scenarios but not the dynamic aspect of the system.

In case of complexity assessment from class diagrams, the following metrics are not only useful to estimate complexity of the system, but may also be used to get a view of its quality. The metrics includes:

- The weighted number of responsibilities of a class both inherited and not inherited.
- The weighted number of dependencies
- Depth of inheritance tree
- Number of immediate sub classes of a class

There are many assumptions in this complexity measuring method and the constants used are based on subjective judgement.

The following table summarises the important points of the three complexity assessment approaches based on three different UML diagrams

Table 1: Summary of three complexity measurement methods using UML diagrams

UML diagram used for complexity assessment	**Objective**	**Parameters considered**	**Diagram Type**	**Assessment Type**	**Characteristics**
Use Case diagram	Early estimate of complexity of the system	Use cases, actors and communications	Static	Uses subjective assessment	1: Involves simple mathematical calculations. 2: Do not consider testing aspect in complexity measurement. 3: Covers Use case view.
Class diagram	Measure the expected complexity and quality of single class	Class responsibilities, dependencies, number of subclasses	Static	No subjective assessment	1: Involves complex mathematical calculations. 2: Do not consider testing aspect in complexity measurement. 3: Covers logical view
Activity diagram	Compute a quantitative measure of complexity	Activities and connections	Dyna mic	No subjective assessment	1: Very simple calculations 2: Considers testing effort in complexity measurement. 3: Covers Use case and process view.

From the summary table it can be concluded that complexity measure using the activity diagram can give a better estimate of complexity as it considers the testing effort by estimating the linearly independent paths from the activity flow graph constructed from the activity diagram. The independent paths can be used for black box testing to test the paths involving the activities in the path. Test cases need to be designed to test the flow that covers the activities in each path.

IV. METHODOLOGY

The proposed approach intends to calculate the coding complexity aspect from the activity diagram representation of the system/class/function without going into the coding aspect contrary to other complexity measures proposed by McCabe, Halstead or Zage. This complexity measure can be considered as a significant indicator for analysing and implementing the architectural aspects related to quality. This measure can be an indicator for estimating the time and effort to build the system. Activity diagrams are used to represent the dynamic aspect of the system. Study on the mapping of UML diagrams with the 4 + 1 view of architecture show that there are different possible mappings proposed in different works [29]. Mapping proposed in the whitepaper “Applying 4+1 View Architecture with UML 2” [30] is considered for this work where activity diagrams are shown to represent the process view of the system. Activity diagrams are constructed to represent the activities either of a class or the system as a whole. In cases if the activity diagram represents the activities of a class, the system complexity is calculated by the aggregation of the individual class complexities of the system. In cases where the activity diagram represents the activities of the system, the complexity value calculated represents the system complexity. The proposed approach starts with the activity diagram, transforms the diagram to a flow graph and then assesses the complexity. To draw the flow graph, termed as Activity Flow Graph (AFG) the two elements – activities and conditions from the activity diagram are considered to be represented in the form of a graph. The graphical representation of the activity diagram is constructed with nodes and connections between the nodes. Every activity and decision point in the activity diagram is represented as a node in the graph. The start and the termination point of the activity diagram are also represented as nodes in the graph. The connection between the activities and the decision points in the diagram is activity diagram is represented by connecting the respective nodes in the graph.

Every connection is considered as an edge in the diagram. A node is represented as a circle and edges are represented as directed arrows. Once the AFG is constructed from the activity diagram, the two metrics required to estimate the complexity are determined.
The following two metrics are considered:

i) Number of nodes and
ii) Number of linearly independent paths.

The number of nodes metrics is chosen as it represents the number of activities and conditions that are directly related to coding complexity in terms of Lines of Code (LOC). More activities and conditions represents increased functionality and decision making points which require more effort in coding and leads to increase in LOC. This increase in LOC further gives rise to greater possibility of errors, defects and faults. Moreover, testing effort will also increase due to more LOC in connection to white-box testing. The second metrics that is the number of linearly independent paths introduces complexity from the testing point of view as more test cases needs to be generated to test each path. Here the testing is related to the black box testing aspect as each path will point to a set of actions/task and decision to reach the termination point.
The proposed complexity measure is defined as :

C= N + IP ----------------------- (1)

Where,
C is the complexity measure,
N is the no. of nodes and
IP is the no. of linearly independent paths.

As defined earlier in software engineering there are three ways to calculate the number of linearly independent paths.

i) Number of edges – Number of nodes + 2
ii) Number of predicate nodes + 1
iii) Number of regions

For our purpose we have considered the method that uses nodes and edges.

IP = E –N + 2 ------------------ (2)

Where,
E is the no. of edges and
N is the number of nodes

A Case study is given here to illustrate the conversion of activity diagram into an AFG and then calculate the complexity measure C from the metrics values collected from the graph.

CASE STUDY: Activity diagram of an order management system converted to corresponding Activity Flow Graph and value of C calculated.

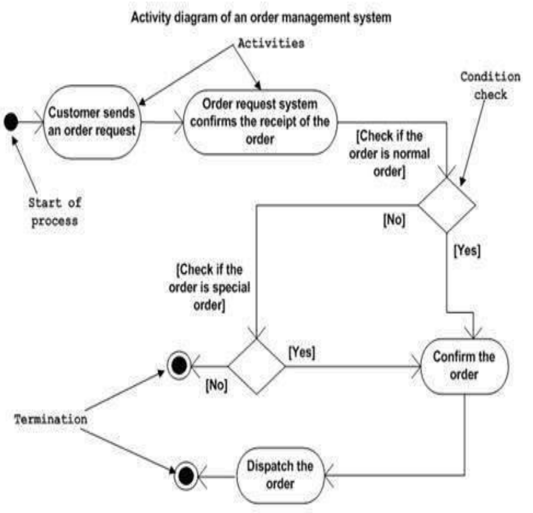
Activity diagram of an order management system
Activities
Condition check
Customer sends an order request
Order request system confirms the receipt of the order
[Check if the order is normal order]
Start of process
[No]
[Yes]
[Check if the order is special order]
[Yes]
[No]
Confirm the order
Termination
Dispatch the order

Fig. 1 Activity diagram
Source: uml-diagrams.org

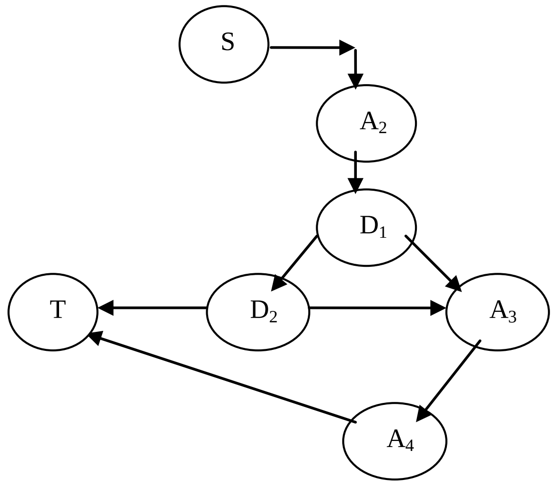

Fig. 2 AFG for Fig.1

For the given case:
AFG notations for the activity diagram are:
Node S represents the start of the activity .
Nodes A_1 to A_4 represents the four activities.
Nodes D_1 and D_2 represents the two decision points.
Node T represents the termination of the activity.

From the AFG, it is found:
Number of nodes, N = 6 (Nodes S and T are not considered as they do not introduce any complexity to the system)
Number of Edges E = 9
IP = 9 - 6 +2 =5
C= 6 + 5 = 11

In this case C=11, which reflects the complexity index of software development, calculated from the activity diagram. Every view represents one aspect of the system, and this quantitative measure calculated aims to quantify the amount of complexity in software development from coding as well as testing point of view. Higher value indicates more complexity due to increase in size which results in more effort in cognitive process, coding as well as testing.

V. IMPLEMENTATION OF THE APPROACH ON A REAL-LIFE TOOL

The proposed approach is used to calculate the complexity of a real-life tool called AppBuilder developed at National Informatics Centre*, which basically tries to generate various application objects (e.g. Forms, Menus etc.) from database metadata. There are eight activity diagrams representing the system. The activity diagram for generating struts based form is given here and its corresponding AFG is generated to calculate the complexity measure.

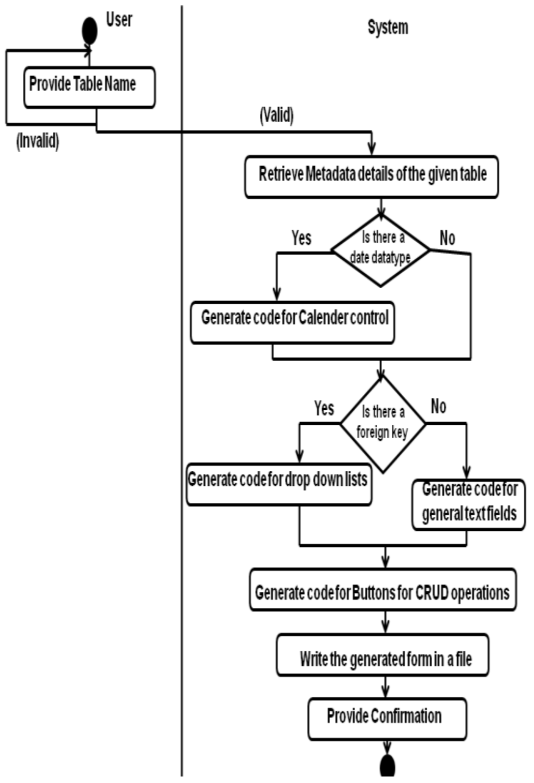
User
System
Provide Table Name
(Valid)
(Invalid)
Retrieve Metadata details of the given table
Yes
Is there a date datatype
No
Generate code for Calender control
Yes
Is there a foreign key
No
Generate code for drop down lists
Generate code for general text fields
Generate code for Buttons for CRUD operations
Write the generated form in a file
Provide Confirmation

Fig. 3. Activity diagram for struts-based forma

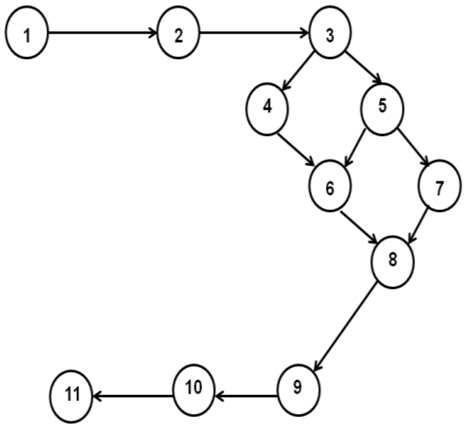

Fig 4. Activity Flow Graph for Fig. 3

The complexity measure for the struts based form is calculated using Eq (1) defined earlier.

No. of nodes (N) = 10

No. of linearly independent paths (IP) = 13 – 10 + 2 = 5

C = 10 + 5 = 15

The C value for all the other activity diagrams are estimated using the proposed approach. The values are given below :

For Javabean class :
No. of nodes = 6.
No. of Edges = 8.
IP = 8 - 6 + 2 = 4.
Therefore, A_2 = 6 + 4 = 10.

For Entity Manager class :
No. of nodes = 11.
No. of Edges = 13.
IP = 13 – 11 + 2 = 4.
Therefore A_3 = 11 + 4 = 15.

For Action class :
No. of nodes = 8.
No. of Edges = 11.
IP = 11 – 8 + 2 = 5.
Therefore A_4 = 8 + 5 = 13.

For configproperties.properties file :
No. of nodes = 7.
No. of Edges = 9.
IP = 9 – 7 + 2 = 4.
Therefore A_5 = 7 + 4 = 11.

For global.properties file :
No. of nodes = 7.
No. of Edges = 9.
IP = 9 – 7 + 2 = 4.
Therefore A_6 = 7 + 4 = 11.

For struts.xml file :
No. of nodes = 6.
No. of Edges = 8.
IP = 8 – 6 + 2 = 4.
Therefore A_7 = 6 + 4 = 10.

For web.xml file :
No. of nodes = 5.
No. of Edges = 7.
IP =7 – 5 + 2 = 4.
Therefore A_8 = 5 + 4 = 9.

The following table represents the C value for the different activity diagrams used in the project.

Table – 2 Calculated C-values from activity diagram

Entity name	C-value
Javabeanclass	10
EntityManagerclass	13
Actionclass	15
Configpropertiesfile	11
Globalpropertiesfile	11
Strutsform	15
Struts	10
Web	9

From table 2 it is observed that the C-value varies from 9 to 15 which signify that there is difference in complexity in case of the listed entities of the system. The creator of the system commenting on the observation concludes that the identified classes having highest C-value of 15 and 13 were indeed the most complex entities to develop. In addition he remarks that, as the project basically tries to generate code for an application in the Model-View-Controller (MVC) architecture, the identified three entities take care of the View (Strutsform), Model (EntityManager) and Controller (Actionclass).

VI. ANALYSIS AND RESULTS

In order to ascertain and obtain a quantitative measure of extent of complexity, further experimentation on the source code of the above mentioned entities was conducted using the SourceMonitor tool. The aim of this experimentation was to explore if there could be a quantitative measure that relates the calculated C-value in terms of existing complexity metrics (LOC). The SourceMonitor tool, version 3.5.3.328 by Campwood is used for this purpose. SourceMonitor is a tool for coders intended for use by programmers as they write code, to expose to them information about the code they are creating. As the project has html, java and xml code, this freeware tool was suitable to estimate the complexity and other parameters like LOC and methods. The table below summarises the different parameter values collected from the analysis..

Table 3: The values of the parameters of the entities

Entity name	Estimated C value (from activity diagrams)	LOC (from tool)	Methods (from tool)	Complexity (from tool)
Javabeanclass(java)	10	56	15	1
EntityManagerclass(java)	13	137	14	14
Actionclass(java)	15	204	14	8
Configfile(java)	11			
Globalfile(java)	11			
Strutsform(html)	15	104	99(tags)	2.9 (complex tags)
Struts(xml)	10	14	8(tags)	0
Web(xml)	9	19	11(tags)	0

As the coding is in different languages, the groups are shown in the table using different colours. Analysis is done based on the two groups - java coded entities and html/xml coded entities. As no parameters are evaluated by the tool for two entities, they are not considered for inference purpose in this study. From the values shown in table 2, it is observed that there is no pattern followed in case of the java class C values and the complexity calculated using the tool. The C values and the LOC values show an increasing trend. Increase in C value shows an increase in LOC. The second part of the analysis is to find a relationship between the C-value and the LOC obtained from the source code using the tool. The table below summarises the analysis to establish a relationship between C and LOC.

Table 4: Relationship of C and LOC

Difference in C-value (X)	Difference in LOC(Y)	Y/X	Relationship of C with LOC(Z)
13-10=3	137-56=81	81/3 = 27	Z=(∑ Y/X)/3 = (27+34+30)/3 = 30.33≈ 30.
15-13=2	204-137=67	67/2 = 33.5 ≈ 34	
15-10=10	204-56=148	148/5= 29.6 ≈ 30	

From Table 4 it can be observed that an increase of one in C value could approximately increase the LOC by 30. It can be inferred that C can be used as an indicator of the size of the system or function or class in case of java code. Other observations in context of html/xml code do not show any significant trend except that a higher value of C implies more number of tags and LOC used for development.

The results concluded after implementing the approach on the activity diagrams of a real life tool and then analysing the source code of the entities can be summarised as follows:

- There is a direct relationship between the C-value calculated using the proposed method with the complexity in developing the system/class/function which the activity diagram represents. Higher C value indicates higher complexity.
- From the observations it is found that, the C-value calculated from the activity diagram can be used to estimate the approximate LOC of the system, class or function for which the activity diagram is drawn.
- Increase in C-value (above 10) by one is equivalent to increase in 30 lines of LOC.

VII. CONCLUSION AND FUTURE SCOPE

The present work on complexity measure proposes a new method for evaluating the process view of the system considering architectural view mapping with UML diagrams. The representation is in the form of an activity diagram representing the process view which is converted to a flow graph. A complexity value is estimated from the representation using two metrics. As one of the metrics measures the independent paths, the complexity measure considers the testing effort into consideration while estimating the complexity value. As an activity diagram can be drawn for a system, class or function, the same approach can be used to estimate the complexity of a system/class/function. This complexity value can be used to estimate the time and effort required to develop the system. The measure of complexity can be used in addition to other attributes like modifiability, operability, maintainability and cost to evaluate the architecture of a system. Implementing the proposed approach on a real life tool, a relationship between the calculated complexity measure and the LOC metrics could be established. This relationship may be used to estimate the size of the system. As size is the basic estimate based on which other project parameters like time, cost and other resources can be estimated, this relationship is significant. The conversion of the activity diagram to AFG provides a basis on which the complexity measure is estimated. This estimation can be further refined to provide more accurate estimates by considering cognitive weights to the nodes and edges. These cognitive weights can consider the subjective aspects in estimation. Further the structural view complexity aspects can be incorporated in refining the complexity measure which is a direction for future work.

REFERENCES

1. Shin, Y., Williams, L.: Is complexity really the enemy of software security?. In: Proceedings of the 4th ACM Workshop on Quality of Protection. ACM (2008)
2. Kruchten,P.," Architectural Blueprints—The "4+1" View Model of Software Architecture", IEEE Software 12 (6),November 1995, pp. 42-50.
3. Weyukar, E.J., " Evaluating Software Complexity Measures", IEEE Transactions On Software Engineering, Vol. 14. No. 9. September 1988, pp.1357-1365.
4. Zuse, H., "Software Complexity, Measures and Methods", vol 4 of Programming Complex Systems, Walter De Gruyetar & Co., Berlin, Newyork, 1991.
5. Isola, E., Olabiyisi, S., Omidiora, E. and Ganiyu, R., "Complexity Metrics and Other Code Based Complexity Metrics", Annals. Computer Science Series. 16th Tome 1st Fasc. – 2018
6. Madhan, M., Anbuarasan, T., Dhivakar, I. and Chandrasegar, T., "Analyzing Complexity Nature Inspired optimization Algorithms using Halstead Metrics", International Conference on Trends in Electronics and Informatics, ICEI 2017.
7. Harrison, W., "An Entropy-Based Measure of Software Complexity", IEEE Transactions On Software Engineering, Vol. 18, No. 11, November 1992.
8. Aboutaleb,H., Monsuez , B., "Measuring Complexity of System/Software Architecture Using Higraph-Based Model", Proceedings of the International MultiConference of Engineers and Computer Scientists 2017 Vol I, IMECS 2017, Hong Kong
9. Polancic, G., Cignar, B., "Complexity metrics for process models – A systematic literature review", Computer Standards & Interfaces, Volume 51, March 2017, Pages 104-117.
10. Kazman, R., Burth, M., "Assessing Architectural Complexity", Proceedings of 2nd Euromicro Working Conference on Software Maintenance And Reengineering (CSMR 98), IEEE Computer Society Press, 1998.
11. Bouwers, E., Lilienthal, C., Visser, J., Deursen , A.,V., "A Cognitive Model for Software Architecture Complexity ",Proceedings of the International Conference on Program Comprehension (ICPC), IEEEComputerSociety, 2010. Software Engineering Research Group Technical Reports: http://www.se.ewi.tudelft.nl/techreports/.
12. Yi, T., Wu,F., Gan, C., "A Comparison of Metrics for UML Class Diagrams", ACM SIGSOFT Software Engineering Notes, Vol 25, Sept'2004.
13. Kang, D., et al. (2004): A structural complexity measure for UML class diagrams. In International Conference on Computational Science (ICCS 2004), Krakow Poland, June 2004, pp.431-435.
14. Kang, D., et al. (2004): A complexity measure for ontology based on UML.In IEEE 10th International Workshop on Future Trends in Distributed Computing Systems (FTDCS 2004), Suzhou, China, May 2004, pp.222-228.
15. Marchesi, M. , "OOA metrics for the unified modeling languages. In Proceedings of 2nd Euromicro Conference on Software Maintenance and Reengineering (CSMR'98), Palazzo degli Affari, Italy, March, 1998,pp.67-73.
16. Sengupta, S., Kanjilal, A., "Measuring Complexity of Component Based Architecture : A Graph based Approach, ACM SIGSOFT Software Engineering Notes, January 2011, DOI: 0.1145/1921532.1921546.
17. Delange, J., Hudak. J., Nichols, W., McHale, J., and Min-Young Nam, "Evaluating and Mitigating the Impact of Complexity in Software Models", Chapter 3, TECHNICAL REPORT ,CMU/SEI-2015 -TR-013, December' 2015.
18. McCabe, T. J. A Complexity Measure. IEEE Transactions on Software Engineering. Volume SE2. 1976. Pages 308–320.
19. Halstead, M. H., " Elements of Software Science", Elsevier North-Holland. 1977.
20. Zage, W. M. & Zage, D. M.," Evaluating Design Metrics on Large-Scale Software", IEEE Software. Volume 10. Number 4. July 1993. Pages 75–81. doi: 10.1109/52.219620.
21. Mesa, J.A., Esparragoza, I., and Maury, H., "Development of a metric to assess the complexity of assembly/disassembly tasks in open architecture products", International Journal of Production Research, DOI 10.1080/00207543.2017.1398431.
22. Anh Nguyen-Duc, "The Impact Of Software Complexity On Cost And Quality – A Comparative Analysis Between Open Source And Proprietary Software', , International Journal on Software Engineering and Application, vol. 8 (2), 17-31, 2017, http://aircconline.com/ijsea/V8N2/8217 ijsea02.pdf
23. Olszewska, M., Dajsuren, Y., Altinger, H., Serebrenik, A., Waldén, M., and Brand, M.G.J.,"Tailoring Complexity Metrics for Simulink Models", ECSA'16 Copenhagen,Denmark, 2016 ACM. ISBN 978-1-4503-4781-5, DOI:10.1145/1235
24. Solodovnikov, A., "Developing Method For Assessing Functional Complexity Of Software Information System", EUREKA: Physics and Engineering, DOI: 10.21303/2461-4262.2016.00157, 2016.
25. Efatmaneshnik, M., and Ryan, M.J., "A General Framework for Measuring System Complexity", Wiley Periodicals, Inc., DOI 10.1002/cplx.21767, 2016.
26. Sedaghatbaf, A., Azgomi, M.A., "A method for dependability evaluation of software Architectures", DOI 10.1007/s00607-017-0568-3, Springer-Verlag GmbH Austria 2017.
27. Sedaghatbaf, A., Azgomi, M.A., "Reliability evaluation of UML/DAM software architectures under parameter uncertainty", IET Software, 2018, Vol. 12 Issue. 3, pp. 236-244
28. Staines, T.S., "Intuitive Mapping of UML 2 Activity Diagrams into Fundamental Modeling Concept Petri Net Diagrams and Colored Petri Nets", IEEE International Conference and Workshop on the Engineering of Computer Based Systems, DOI 10.1109/ECBS.2008.12, 2008.
29. https://softwareengineering.stackexchange.com/questions/233257/mapping-between-41- architectural -view-model-uml accessed on 16/10/17
30. www.fcgss.com, "Applying 4+1 View Architecture with UML 2", White Paper, 2007.