Asian Journal of Computer and Information Systems (ISSN: 2321 – 5658)
Volume 9 – Issue 1, February 2021

Software Functional Complexity Using a New Set of Criteria

Bhaskar Sinha[1], Pradip Peter Dey[2], Mohammad Amin[3]

[1] National University
San Diego, California, USA
Email: bsinha [AT] nu.edu

[2] National University
San Diego, California, USA
Email: pdey [AT] nu.edu

[3] National University
San Diego, California, USA
Email: mamin [AT] nu.edu

ABSTRACT— ***With the rapid technology advances, there is an emerging consensus that the size and complexity of software designs are increasing so rapidly that they proportionally affect the magnitude of administrative and development efforts. An important consideration is how to estimate software complexity. This subject continues to be a research topic in the literature. The software design process researched here uses the Unified Modeling Language (UML) diagrams and the database design for extracting pertinent information. The Entity Relationship (ER) model of Peter Chen (of MIT) is a conceptual method of describing the data in a relational structure. An Entity Relationship Diagram (ERD) and an Entity Relationship Schema (ERS) represents the ER model, containing the entities, attributes, primary and foreign keys, and the relationships between the entities. Extending this ERS modeling construct, this paper uses an additional enhanced schema, called the Object Relationship Schema (ORS), which, together with the existing ERS, creates an enhanced view of the requirements and the design of the database. In addition, functional dependency, security, computational complexity, use cases, component structure and interpretations are considered for estimating functional complexity of modern software systems which is very valuable in higher education for new workforce development.***

Keywords - Complexity index, complexity multiplier, entity relationship model, object relationship model, unified modeling language (UML).

---

1. INTRODUCTION

Ousterhout suggests that writing computer software is one of the purest creative activities in the history of human race [1]. Programming is an imaginative and artistic activity, and the ability to understand this complicated creation is a fascinating challenge. Ousterhout further asserts that as the sophistication of a piece of software increases, designing and implementing new features and extensions becomes more complex, risky, and time-consuming [1]. This leads to an increase in cost associated with further enhancements and, most importantly, cost of maintaining the system. Although many tools are available to deal with this complexity, more detailed understanding of the design aspects that contribute to the software complexity is necessary. Modular design and structured design have helped reduce software and program complexity in the past, and the concepts of Objects, used in Object Oriented Programming Systems (OOPS), has benefited this process significantly. This study suggests that more objective measures of the software design are still a challenge, especially understanding this complexity measure at the design phase before programming and coding starts. Also, the waterfall model for software development, where the design phase is a front-end step in the complete software development life cycle, has been largely replaced by incremental approaches, such as the agile development process. This paradigm shift has made the design phase a continuous process throughout the life cycle of the system development, thus, understanding and standardizing the complexity of the system at the design phase has become even more critical. General approaches to reduce or eliminate complexity in software have been suggested in the literature [2-5]. These include minimization or elimination of special cases in code, modular design, and code encapsulation to enable programming without being exposed to all its complexity all at once. This research is not specifically about reducing or minimizing complexity, but rather, about understanding and estimating the complexity during the development phase of the project.

Since Albrecht's initial proposal about the function-point method in 1979 there has been considerable interest in estimating functional complexity in software [6]. The emergence of functional dependency in databases and the rise of scenario-based methods such as use case scenarios or user stories in software development inspires us to have a fresh look at the topic and propose a new approach by combining multiple criteria including, functional dependency, security, computational complexity, use cases and related issues [7-10]. "Complexity comes from an accumulation of dependencies and obscurities" [1, page 11]. Security is one of the crosscutting concerns that may contribute significantly towards software complexity in most contexts as the general security problem is proven to be un-decidable in the HRU model [11]. A significant portion of expenditures in computing systems development is the maintenance of software. Working with billions of lines of code (LOC) that exist, the software maintenance represents an Information Systems (IS) activity of considerable importance [12]. Resource allocation for maintaining this software must be related to and based on certain aspects of the software. Many methods of software complexity measure have been proposed in the past for this and various other reasons [1,8-25]. Most of these, although giving a good representation of complexity, do not specify a procedure for easy measurement. The McCabe metric [19] and the Halstead's method [16] are two widely referred studies in this area. The McCabe metric uses the number of control edges in the code [19], and Halstead's method uses a mathematical relationship between the variables and the programming statements [16-18]. Others have used code coverage or specification-based methods, such as the function point method [13]. Chidamber and Kemerer [15] introduced a Metrics Suite for object-oriented design and code. Other methods used for measuring software complexity are decisional complexity (McClure Metric) [20], data access complexity (Card Metric), branching complexity (Sneed Metric), and data complexity (Chapin Metric) [14]. This research describes a quantitative method to measure software complexity combining certain aspects of the function point method [13], LOC, coding complexity abstractions, and use-case analysis. The role of iterative development of graphical user interfaces and use of objects such as buttons were not considered in the original function point method [13]. Use case analysis relates functional requirements to use case scenarios in descriptive and visual forms such as use case diagrams [10,25].

Database modeling and design is an ongoing, agile technological revolution that has enabled us to work with vast amounts of evolving types of data. Codd introduced the relational data model in 1970 [22-23]. Chen later proposed the ER model for intuitive and conceptual dimensions of data [24]. Objects, as intended for this paper, include traditional well-structured data; unstructured data; aggregate data such as forms, SQL queries and reports; graphics; audio; video; and Binary Large Objects (BLOBs) [2,26]. The growth of the complexity of database content transfers complexity to the system development process. Consequently, data complexity impacts the entire computing infrastructure: development, processing, security, maintenance, transmission, viewing and archival for a beginning list. Methods have been proposed in literature to estimate the complexity of a database. Specification-based methods, such as the function point method [13] have been proposed in literature. Zuse introduced the theoretical validation of a software measurement theory-based framework [27] and Briand introduced the axiomatic based framework [28]. Another research performed statistical analysis and concluded that the number of foreign keys in a database design is a better indicator of understanding the schema than the depth of the referential tree [29-30]. A method to estimate this complexity, called the DC (Database Complexity), uses the database indicators like attributes, keys, indices, and database references that are readily available in a Relational Database Management System (RDBMS) to quickly determine the complexity [29-30]. Other methods considered not only the tables and relationships in the database schema, but also an application layer consisting of queries, forms, and reports [26,30]. The present research builds on the cited past work and proposes the complexity introduced by the database as a quantitative technique for estimating database complexity based on information available from the ERS and the Object Relational Schema (ORS) [2]. To assist developers and programmers achieve the optimized design, an intuitive and conceptual method of creating an abstract database model is the ERD that leads to the schema - the ERS. These describe the relationships between the entities in the system and are implemented in the form of tables with keys that help in conceptualization of the relations. With increased demands for the development of a database project to be agile and meet aggressive schedules, various aspects of this model are periodically reviewed for possible improvements and accordingly enhanced.

Based on the ERS modeling technique, Sinha, et. al. [31] have suggested creating a superset of the ERS that specifies the relations between all objects in the systems. The collection of tables specifying the entities in an ERS is one object of this model, described as the table object. Other objects in this model included in this research are the, forms, queries, and the reports that are the requirements of a design. Relations are specified among objects in the same manner as for the ERD, and from these relations, a superset of the ERS is created. This construct, termed the Object Relationship Schema (ORS), shows the relationships between all objects of the database that are interdependent and need to be implemented as a complete single interconnected system. Like the ERS method of implementing entities as tables, this research [31] introduced the implementation of objects as clusters in the Object Relationship Diagram (ORD). It is claimed that this new ORD construct, together with the existing ERS, will improve the capability and usability of RDBMS modeling and database development significantly. This combination of ERS and ORS also provides a better understanding of the complete database requirements and specifications, thus improving the team's ability to plan for and allocate appropriate resources for development, maintenance, and implementing enhancements during the operational lifespan of this product. The database complexity, described in the following sections, considers all Objects mentioned previously and focuses on the aggregate data types of Forms, SQL Queries and Reports [4-5].

2. SOFTWARE COMPLEXITY

This research describes a quantitative method to measure software complexity combining certain aspects of the function point method [13], lines of code (LOC), "coding complexity abstractions", and "use case diagrams" [14,20-25]. The role of iterative development of GUIs and use of objects such as buttons were not considered in the original function point method [6]. Use case diagram is one of the most popular UML diagrams in software development [25]. Use case are usually developed before the coding phase and therefore the software complexity can be estimated at an early stage of the development process enabling us to manage risks in subsequent phases. Factors used for estimating the Unadjusted Complexity Index (UCI) and predefined weight assigned for each functional item is shown in Table 1.

CRITERIA FOR (UCI) ESTIMATION	**RECOMMENDED POINTS**			
INPUTS/OUTPUTS	**SIMPLE**	**AVERAGE**	**COMPLEX**	**SCORES**
Number of Inputs	3 (each)	4 (each)	6 (each)	
Number of Outputs	4 (each)	5 (each)	7 (each)	
COMPONENTS				
Number of Interactive Components (5 or less)	2 (each)	5 (each)	9 (each)	
Number of Interactive Components (6 to 20)	10 (each)	20 (each)	30 (each)	
Number of Interactive Components (21 or more)	40 (each)	50 (each)	80 (each)	
USE CASE DIAGRAM				
Number of Use Cases	3 (each)	5 (each)	10 (each)	
Number of Actors	2 (each)	5 (each)	10 (each)	
Total Number of Interfaces	5 (each)	10 (each)	20 (each)	
CLASS DIAGRAM				
Total Number of Classes	2 (each)	4 (each)	8 (each)	
Total Number of Relationships Among Classes	2 (each)	4 (each)	8 (each)	
Total Number of Functions	3 (each)	6 (each)	20 (each)	
DATABASE: ENTITY RELATIONSHIP SCHEMA				
Number of Entities	2 (each)	10 (each)	20 (each)	
Number of Tables	5 (each)	10 (each)	20 (each)	
Number of Attributes	2 (each)	3 (each)	4 (each)	
Number of Foreign Keys	2 (each)	3 (each)	4 (each)	
Number of Dependencies	2 (each)	4 (each)	10 (each)	
DATABASE: OBJECT RELATIONSHIP SCHEMA				
Number of Objects	3 (each)	5 (each)	8 (each)	
Number of Clusters	5 (each)	10 (each)	20 (each)	
SECURITY LEVEL				
Level 1 Security (username, password etc.)	5	10	20	
Level 2 Security (zero trust)	50	100	200	
TOTAL: UNADJUSTED COMPLEXITY INDEX (UCI)				**TOTAL**

Table 1. Unadjusted Complexity Index (UCI) Measurement Criteria

The Entity Relationship Schema (ERS) of a relational database design maps the relationships among all the entities of the database. In a relational database, functional dependencies and their rolls are very important. If these issues are not addressed at the initial stage of the design process, then the database will generate inaccurate information. Removal process of functional dependency is called normalization and it increases the number of tables in the database. A properly normalized database increases the implantation complexity and requires a more advanced database engine in the RDBMS. The traditional ER model with the ERS does not contain the relationships of these entities with the other Objects of this single system. The additional part in the design specifies requirements of the other Objects in the system - the Forms, SQL Queries, and the Reports. The design complexities of creating these Objects are subjective in the traditional ER model. The ORS is the entire view of the required specifications, where associations and relations between all Objects of this interconnected system are captured [2].

Based on the ERS modeling technique this augmented schema, termed the ORS, is created which, together with the ERS, creates a complete view of the design requirements of the database. The ORS captures the relationships between all Objects of the database that are interdependent and need to work as a single complete unit. Like the ERS method of implementing entities as tables, the ORS uses the concept of implementing Objects as clusters, thus creating an expanded set or superset of the ERS. The combination of the existing ERS construct and this new ORS model together will significantly enhance the capability of the RDBMS by providing a better understanding of the database and design requirements. This will improve the development process and enable an efficient implementation of the project. The complexity measure introduced by the database, is estimated at the design stage and this is done using the ERS, the ORS, and some suggestions from function point research [13]. The criteria for this determination are shown in Table 1. Contributions to the total database complexity count are from the ERS and the ORS. The ERS provides the number of entities, number of tables, and information from each table. The ORS provides the number of objects, number of clusters, and information from the content of each cluster. It is reasonable to use some influence factors like ones suggested by Albrecht [6,11]. The 14 influence factors, formulated as questions with values of each ranging from 0-5, and their use in adjusting and determining the Complexity Index (CI) are proposed in Table 3.

To clarify how Table 1 and Table 3 work together for the Complexity Index of an application, consider a small example problem: *A computerized system needs to be built for a bicycle rental business. A rental office in a California beach area lends bicycles of different types. The assortment of bicycles comprises Racing bikes, Mountain bikes, Touring bikes, and Specials. Three types of Mountain bikes are: (1) Downhill, (2) Freeride, and (3) Cross Country. Three types of Specials are (a) Triplets, (b) Tricycles, and (c) Kids bikes. A client may reserve a bike of a certain category for a certain period. The reservation can be guaranteed by using a credit card. The rental office guarantees that a bike of the desired category will be available for the requested period according to the reservation. The client can request changes to the reservation any time before the rental contract is signed, subject to certain constraints. Bikes are available at hourly rates, daily rates and weekly rates, no extra charge for helmets etc. When the client fetches the bikes, he or she must sign a rental contract. Bikes for kids are available with parental contracts. Within the reserved period or immediately thereafter the client returns the bikes and pays the bill.*

Based on the above bike rental problem, initials requirements analysis and design are performed. The following use case diagram in Figure 1 identifies three types of users shown outside the box that includes major use cases as ovals; each use case is based on a story about how a user interacts with the system [10].

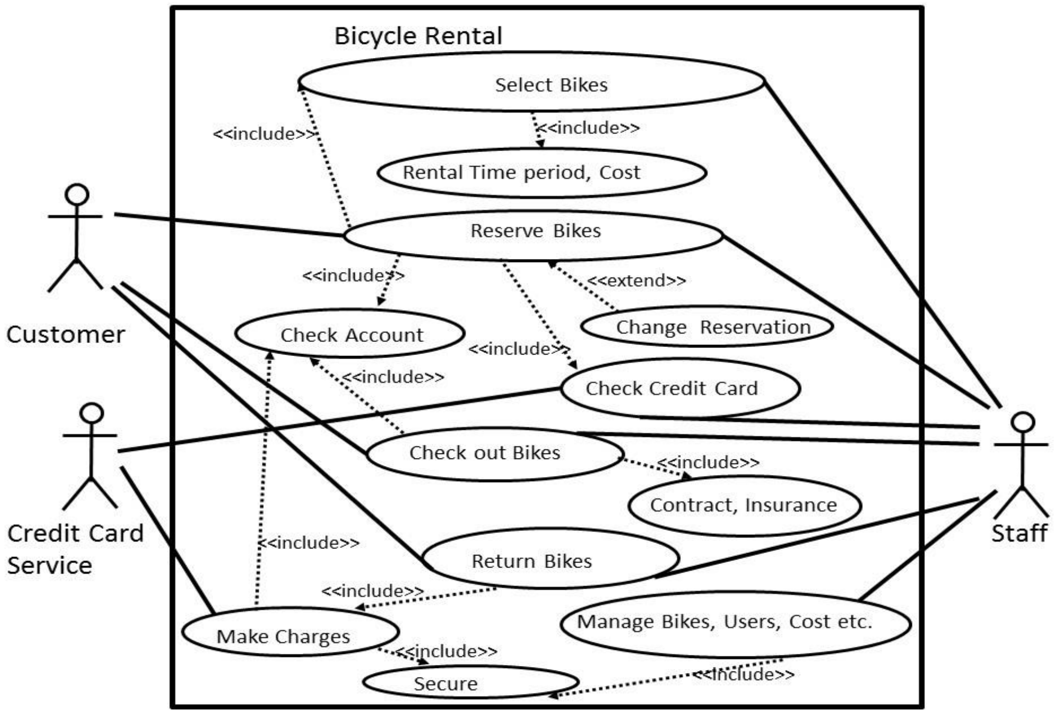

Figure 1. Use-Case Diagram

Based on the development strategies for the bicycle rental system, Table 2 and Table 3 are prepared for the Complexity Index calculations.

CRITERIA FOR (UCI) ESTIMATION	**RECOMMENDED POINTS**			
INPUTS/OUTPUTS	**SIMPLE**	**AVERAGE**	**COMPLEX**	**SCORES**
Number of Inputs	3 (each)	4(each) 10*4	6 (each) 4*6	64
Number of Outputs	4 (each)	5 (each) 4*5	7 (each) 3*7	41
COMPONENTS				
Number of Interactive Components (5 or less)	2 (each)	5 (each)	9 (each)	
Number of Interactive Components (6 to 20)	10 (each)	20 (each) 8*20	30 (each) 4*30	280
Number of Interactive Components (21 or more)	40 (each)	50 (each)	80 (each)	
USE CASE DIAGRAM				
Number of Use Cases	3 (each)	5 (each) 9*5	10 (each) 3*10	75
Number of Actors	2 (each)	5 (each) 3*5	10 (each)	15
Total Number of Interfaces	5 (each) 2*5	10 (each) 8*10	20 (each) 4*20	170
CLASS DIAGRAM				
Total Number of Classes	2 (each) 2*2	4 (each) 20*4	8 (each) 10*8	164
Total Number of Relationships Among Classes	2 (each)	4 (each) 12*4	8 (each) 2*8	64
Total Number of Functions	3 (each) 10*3	6 (each) 90*6	20 (each) 8*20	730
DATABASE: ENTITY RELATIONSHIP SCHEMA				
Number of Entities	2 (each) 10*2	10 (each) 15*10	20 (each)	170
Number of Tables	5 (each) 2*5	10 (each) 6*10	20 (each)	70
Number of Attributes	2 (each) 12*2	3 (each) 3*3	4 (each)	33
Number of Foreign Keys	2 (each)	3 (each) 2*3	4 (each)	6
Number of Dependencies	2 (each) 3*2	4 (each) 4*4	10 (each)	22
DATABASE: OBJECT RELATIONSHIP SCHEMA				
Number of Objects	3 (each) 520*3	5 (each)	8 (each)	1560
Number of Clusters	5 (each) 5*5	10 (each)	20 (each)	25
SECURITY LEVEL				
Level 1 Security (username, password etc.)	5	10	20	
Level 2 Security (zero trust)	50	100 4*100	200	400
TOTAL: UNADJUSTED COMPLEXITY INDEX (UCI)				**3889**

Table 2. Unadjusted Complexity Index (UCI) Measurement Calculations

INFLUENCE FACTOR BASED QUESTIONS	INFLUENCE (0-5)
1. Does the system require reliable backup and recovery?	5
2. Are there complex interactions among components?	4
3. Are there components that are expected to work in an autonomous mode?	4
4. Are there components with intractable problems?	0
5. Are there security issues with no known algorithmic solution?	4
6. Does on-line data entry require input transaction to be built over multiple screens or operations?	2
7. Is the system required for safety-critical actions?	1
8. Are the input, outputs, files, or inquiries complex?	3
9. Is the internal processing complex?	2
10. Is the code required to be reusable?	3
11. Is the system designed for multiple installations in different locations?	2
12. Is the system required to deal with massive amounts of data?	1
13. Is the system designed to facilitate change and ease of use by the user?	4
14. Any other factor: (Explanations: Accidents, Exceptions, Contract violations etc.)	4
Sum of Influence factors	39
Complexity Multiplier = 0.65 + 0.01*(Sum of Influence Factors)	1.04
Complexity Index (CI) = Complexity Multiplier * UCI	**4045**

Table 3. Complexity Index Factors and Calculations

The Complexity Index (CI) for the bike rental problem described above is 4045, which is obtained by multiplying the UCI, 3889, from Table 2 with 1.04 (Complexity Multiplier) as indicated in Table 3. Note that factors in Table 3 are mainly based on Albrecht [3,13]. Factor number 14 in Table 3 is a new factor that allows "Any other factor" with explanations, to provide flexibility to participant developers; often developers face compliance problems, social issues, regulations, project specific issues, etc. that could be included in this factor. The CI (4045) is based on our work at the early development stage; it is likely to change on reflection from iterative development phases. Before calculating the complexity value, initial requirements analysis is done, use cases are reviewed, and a use case diagram is drawn. Use cases (or user stories) play an important role in our software complexity analysis; however, use case points method is narrowly defined [32] and inadequate for many software characteristics considered as a new set of criteria in this study. The bike rental system is a relatively simple problem chosen to explain how the criteria that are combined based on Albrecht's pioneering work [6,13]. In Table 1 and 2, modern database related concepts have replaced Albrecht's count of logical files. One of the most important issues is the contribution of computer system security issues towards software complexity that interacts in many complicated ways with other aspects of software requirements, design, and development. Albrecht's function point method was appropriate for functional size of software in the past decades [10]. We emphasize functional complexity (not size) as an extended approach, because use case analysis, database dependency, computer system security, GUI etc. go beyond size-oriented approaches and properly estimate software functional complexity aspects. For other alternative views and perspectives, you are encouraged to consider suggestions from International Function Point Users Group (IFPUG) [33].

5. CONCLUSIONS

A measurement technique for software complexity is described in this paper which was motivated and influenced by the function point method [13]. This technique, using a new set of criteria, defines the two terms, the Unadjusted Complexity Index (UCI) and the Complexity Index (CI), that are derived and justified by reasoning of use case diagrams combined with intuitive idea of code building block abstractions and GUI efforts. This method may also be used to estimate the impact of software complexity on the costs of software maintenance projects in IS environments. For the database design, the complexity introduced by the data design is an extension of earlier work in this area [3,31,34]. The method is adequate for estimating the complexity for a wide variety of systems with database components from their design and development characteristics. This method is valuable in higher education for the new workforce development required by the 21st century software industry; it is based on a broad spectrum of criteria. We hope that most software engineers will be receptive to the broad set of criteria used in this paper because software engineering needs to deal with these aspects in practice. Suggestions for future research include considerations of social issues, human cognition, functional dependency types, metadata, comprehension strategies, patterns and organization of semi-structured and unstructured databases, and other emerging strategies in software development. Social issues related to COVID-19 pandemic may be of special consideration for future research. This will provide a better estimation of the complete system requirements and specifications, thus improving the ability to plan for and allocate appropriate resources for development and maintenance of the software systems during the operational life span. This current paper emphasizes conceptual clarity about functional complexity setting the stage for future work that may demonstrate methodological details with examples from various domains. We hope to see more advanced work in this area during the next few years.

6. ACKNOWLEDGEMENTS

As a group of National University faculty, the authors thank and gratefully acknowledge the help and support received from the administration, staff, and other faculty members in the Department of Engineering and Computing, College of Professional Studies, at National University, during the continuing research on this topic and the preparation of this paper.

7. REFERENCES

[1] Ousterhout, J. A. Philosophy of Software Design. Yaknyam Press, Palo Alto, California, USA. 2018.

[2] Sinha, B., Dey, P., Amin, M. and Romney, G. Database Modeling with Object Relationship Schema. 12th International Conference on Information Technology Based Higher Education and Training, Antalya, Turkey, IEEE Xplore 10.1109/ITHET.2013.6671029, October 2013.

[3] Sinha, B., Dey, P., Amin, A., and Badkoobehi, H. Software Complexity Measurement Using Multiple Criteria. *Journal of Computing Sciences in Colleges (JCSC)*, ISSN: 1973-4771 print, 1937-4763 digital, Vol. 28(4), Pages 155-162. April 2013.

[4] Sinha, B., Romney, G., Dey, P., and Amin, M. Estimation of Database Complexity from Modeling Schemas. *Journal of Computing Sciences in Colleges (JCSC)*, ISSN: 1973-4771 Vol. 30(2), Pages 95-104. http://dl.acm.org/citation.cfm?id=2667432&picked=prox&cfid=624613975&cftoken=60959621, December 2014.

[5] Sinha, B., Romney, G., Bowen, D., Dey, P., and Amin, M. Augmenting Entity Relationship Schema for a Complete View of a Relational Database. Proceedings of the *4th International Joint Conference on Advances in Engineering and Technology (AET-2013) Track: Advances in Computer Science (AET-ACS 2013*), *@Elsevier 2013,* Gurgaon, Delhi-NCR, India, December 13-14, 2013, Pages 462-469. http://aet.theides.org/201. December 2013.

[6] Albrecht, A. Measuring Application Development Productivity, Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, Monterey, California, October 14–17, IBM, pp. 83–92, 1979.

[7] Fischman, L. Evolving Function Points, Crosstalk: The Journal of Defense Software Engineering. 2001. Volume 14, Issue 2. pp 24-27. Retrieved on July 2, 2018, from: http://static1.1.sqspcdn.com/static/f/702523/9452310/1289969383637/200102-Fischman.pdf?token=KtIfYtLECcKOXlTCGQyFmwpQJwE%3D

[8] Sipser, M. Introduction to the Theory of Computation, 3rd edition, Cengage Learning, 2012.

[9] Kozaczynski, W. (Voytek) and Ning, Jim Q. Component-Based Software Engineering (CBSE), Proceedings of the 4th International Conference on Software Reuse (ICSR '96), Retrieved on July 2, 2018, from: https://www.computer.org/csdl/proceedings/icsr/1996/7301/00/73010236.pdf

[10] Pressman, R. and Maxim, B. Software Engineering: A practitioner’s Approach. (8th Edition), McGraw-Hill, 2015

[11] Harrison, M., Ruzzo, W., and Ullman, J. Protection in Operating Systems, Communications of the ACM. 19 (8): 461–471, 1976.

[12] Banker, R. D. Software Complexity and Maintenance Costs, *Communications of the ACM,* 1993.

[13] Albrecht, A. and Gaffney, Jr., J. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation, *IEEE transactions on Software Engineering*, Vol.9 Issue 6, Pages 639-648, 1983.

[14] Chapin, N. A. Measure of Software Complexity, Proceedings of the 1979 National Computer Conference, New York, pages 995-1002, 1979.

[15] Chidamber, S. R. and Kemerer, C. F. A Metrics Suite for Object Oriented Design, *IEEE Transactions on Software Engineering,* Vol. 20(6), 476-493, 1994.

[16] Halstead, M. H. Elements of Software Science, Elsevier North-Holland, 1977.

[17] Kearney, J. K., Sedlmeyer, R. L., Thompson, W. B., Gray, M. A., and Adler, M. A. Software Complexity Measurement, *Communications of the ACM*, Volume 29, Number 11, 1986.

[18] Marco, L. Measuring Software Complexity, *Enterprise Systems Journal*, April 1997.

[19] McCabe, T. J. Software Complexity, *IEEE Transactions on Software Engineering.* P.308-320, Issue 4, Volume SE-2, December 1976.

[20] McClure, C. L. A Model for Program Complexity Analysis, *Proceedings of the 3rd international conference on Software engineering,* p.149-157, May 10-12, 1978, Atlanta, Georgia, United States, 1978.

[21] Weyuker, E. J. Evaluating Software Complexity Measures, *IEEE Transactions on Software Engineering,* Vol. 14. No. 9, 1988.

[22] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM (CACM). Vol.13, Pages: 377-387, June 1970.

[23] Codd, E. F. and Rustin, In R. (ed.): Relational Completeness of Data Base Sublanguages. *65-98, Prentice Hall and IBM Research Report RJ 987*, San Jose, California March 1972.

[24] Chen, P. The Entity Relationship Model - Towards a Unified View of Data. *ACM TODS*, Vol.1, No.1, 1976.

[25] Booch, G. Unified Modeling Language User Guide, (2nd Edition). Addison-Wesley, 2017.

[26] BLOB Binary Large Object, from http://docs.oracle.com/javadb/10.8.2.2/ref/rrefblob.html May 22, 2014.

[27] Zuse, H. A. Framework of Software Measurement. Publisher: De Gruyter Incorporated, Walter XXIX, 755 pages. ISBN-10: 3110155877 ISBN-13: 9783110155877, 1998.

[28] Briand, L. C., Morasca, S., and Basili, V. R. Property-based software engineering measurement, IEEE Transactions on Software Engineering, 22 (1), 68-86, 1996.

[29] Piattini, M., Calero, C., and Genero, M. Table Oriented Metrics for Relational Databases. *Software Quality Journal*, 9(2), 79-97, 2001.

[30] Pavlic, M., Kaluza, M., and Vrcek, N. Database Complexity Measuring Method. Central European Conference on Information and Intelligent Systems, CECIIS 2008.

[31] Sinha, B., Romney, G., Bowen, D., Dey, P., and Amin, M. Relating Objects to Relational Database Design. *ACEEE International Journal on Information Technology*. Publisher: ACEEE, USA (a division of IDES). ISSN 2158-012X (print); ISSN 2158-0138 (online), Vol.5 Issue 1, Pages 1-8. http://journals.theaceee.org/, September 2015

[32] Wikipedia. Use Case Points. Retrieved December 2020, https://ed.wikipedia.org/wiki/Use_case_points.

[33] International Function Point Users Group. https://www.ifpug.org/

[34] Jamil, B. and Batool. A. SMARtS: Software Metric Analyzer for Relational Database Systems. *Information and Emerging Technologies (ICIET) 2010 International Conference*, June 2010.