International Journal of Computer Science Engineering & Technology
APPROVED BY
NATIONAL SCIENCE LIBRARY (NSL)
NATIONAL INSTITUTE OF SCIENCE COMMUNICATION AND INFORMATION RESOURCES (NISCAIR)
COUNCIL OF SCIENTIFIC AND INDUSTRIAL RESEARCH (CSIR)- NEW DELHI INDIA .
ISSN :2455-9091

COMPARISON OF SPATIAL COMPLEXITY WITH SOFTWARE ENGINEERING MODELS

[1] M. SUBASH, [2] MS. D. KAVITHA
[1] Research Scholar, [2] Head of the Department,
[1, 2] Department of Computer Science,
[1, 2] KGISL Inst. of Information MGT , TamilNadu, India.

ABSTRACT- Software Engineering is worried about designing, writing, testing, implementing and maintaining software. It frames the premise of operational design and development to all computer systems. In this paper presents a bunch of straightforward software complexity metrics that has been roused by improvements inside cognitive brain research. Complexity measures are developed by breaking down the distance between segments of a program. The more prominent the distance between program pieces, the more noteworthy the subsequent spatial complexity of a program. Recommendations are made with respect to how spatial complexity measures can be custom-made to singular programmer groups. Utilizing these metrics, the complexity of a software framework can be changed utilizing abstract measures of programmer experience and information.

Keywords – [Spatial Complexity, Procedure-Oriented Software, Software Metrics, Psychological Complexity, Software Comprehension, Software Engineering, waterfall model, SDLC.]

1. INTRODUCTION

Software maintenance and for the most part software comprehension represent the biggest expenses in the software lifecycle. To evaluate the expense of software comprehension, different complexity measures have been proposed in the literature Intelligence tests analyze various cognitive abilities. Verbal capacity is tried. Graphical and textual based tests are utilized to test enlistment, and spatial abilities are tried utilizing mental rotation tasks. Spatial capacity has been related with the selection of critical thinking strategy, and has assumed a significant part in the definition of a persuasive model of working memory. To effectively tackle debugging, maintenance and comprehension tasks, programmers should have information on the programming language, have a comprehension of the application domain and build up an enthusiasm for the connections that can exist between the two. To build up a comprehension of non-trifling software systems, a programmer should start to know where huge parts of the program lie and have an enthusiasm for their significance to different parts of a program. Spatial capacity has been related with the selection of critical thinking strategy, and has assumed a significant part in the definition of a persuasive model of working memory.

Software isn't just encoded within a solitary source file yet can be disseminated among quite a few different files. The possibility of the programming plan or program schema has

been utilized as an explanatory device to clarify programmer ability. Letovsky and Soloway accepted that programming plans can be arranged within various parts of a program, and this can make programs hard to comprehend.

Spatial complexity is characterized here as the trouble to work on the structure or type of a 2-and-higher-dimensional surface or object. Spatial complexity ought not to be mistaken for "space complexity", "topological complexity", "shape complexity" or "complex systems".

Figure 1. Spatial Complexity

Spatial complexity assumes a significant part in developing the viable software. One of the significant exercises of the maintenance stage is to comprehend the source code first, and on the off chance that any progressions are needed in source-code, The correlation between the orientation and area of different substances with their taking care of ought to be set up by the developers, which requires spatial capacities.

Spatial Complexity Metrics

Intelligence tests look at various cognitive abilities. Verbal capacity is tried. Graphical and textual based tests are utilized to test enlistment, and spatial abilities are tried utilizing mental rotation tasks. Spatial capacity is a term that is utilized to allude to a person's cognitive abilities identifying with orientation, the location of objects in space, and the handling of location related visual data.

To effectively address debugging, maintenance and comprehension tasks, programmers should have information on the programming language, have a comprehension of the application space and develop an enthusiasm for the associations that can exist between the two. To build up a comprehension of non-insignificant software systems, a programmer should start to know where huge parts of the program lie and have an appreciation of their significance to different parts of a program.

Software Engineering

Software engineering is a piece of computer science that incorporates the turn of events and working of computer systems software and applications software.

Planning: Planning for the quality affirmation prerequisites and ID of the risks related with the undertaking is likewise done in the planning stage.

Modeling: Models are types of depiction frequently embraced in software development. They are reflections used to address and convey what is significant, without superfluous detail, and to help developers manage the complexity of the issue being.

Construction: Software construction is a software engineering discipline. It is the nitty gritty formation of working significant software through a blend of coding, verification, unit testing, integration testing, and debugging.

Deployment: When the product is tried and fit to be conveyed it is delivered officially in the proper market. At times product deployment occurs in stages according to the business strategy of that organization. At that point dependent on the feedback, the product might be delivered for what it's worth or with recommended enhancements in the focusing on market segment.

Communication: Extraordinary communication is the main trademark for progress as a software engineer. Our work regularly includes working across numerous parts of the organization speaking with product management, account management, support, operations, sales, customers and obviously with our friends and managers.

Software Process Models

A Process Model describes the sequence of stages for the whole lifetime of a product. Consequently it is some of the time additionally called Product Life Cycle. It presents a description of a process from some particular viewpoint as:

Specification: Software specification or requirements management is the process of understanding and defining what functional and non-functional requirements are needed for the system and identifying the limitations on the system's activity and development.

Design: A design model in software engineering is an object-based picture or pictures that address the utilization cases for a system.

Validation: Validation is the process to assess the software after the fulfillment of the development stage to determine whether software meets the client assumptions also, requirements.

Evolution: The evolutionary model is a mix of the Iterative and Incremental model of the software development life cycle. The Evolutionary development model divides the development cycle into more modest, incremental waterfall models in which clients can gain admittance to the product toward the finish of each cycle.

Software Development Life Cycle

There are different software developments life cycles models characterized and designed which are followed during the software development process. These models are additionally eluded as "Software Development Process Models". A Programming process model is a theoretical portrayal to depict the process from a particular perspective. There are amounts of general models for software processes, and so forth this exploration will see the accompanying five models:

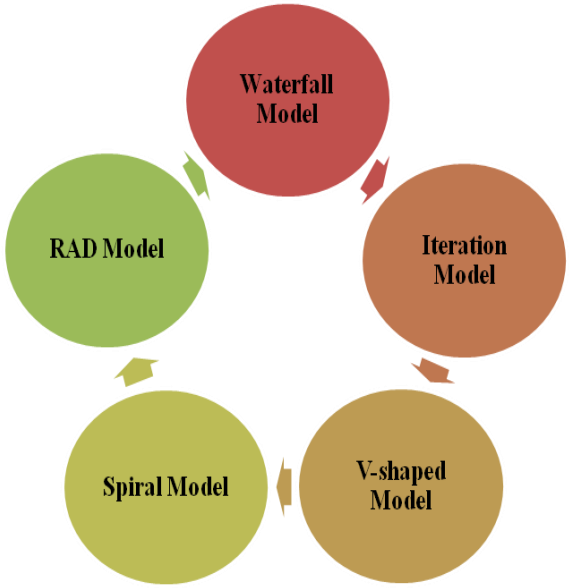

Figure 2.SDLC Model

Problem Statement

The issues distinguished in this exploration are for Spatial Complexity and Software Engineering Process Model in development programs reason. The exploration issues distinguished are featured underneath:

- With most undertakings, this cycle rehashes basically a similar way - from the outset there is an aggressive arrangement, at that point something unforeseen occurs, and afterward all decisions are tossed into the waste.
- This doesn't look so startling with regards to a little project. In any case, now and again we simply need to concede, regardless of the amount it torments us that it's ideal to change a multi-million dollar project without any preparation than to interminably keep in a coma and fix the bugs that will inevitably reoccur.
- Also called computational complexity, algorithmic complexity implies the capacity relies upon the size of the input data and outputs the amount of work done by a specific

algorithm. The amount of work, for this situation, is generally estimated by dynamic ideas of existence, which are designated "computational resources." The speed of development diminishes as the undertaking develops.

- How does this issue influence work? In the first place, as we have effectively referenced, it is difficult to accurately anticipate the circumstance. Also, technology is quickly evolving. Thirdly, the complexity of development is the explanation that ventures need an ever increasing number of programmers, however they do less and less work.

2. EXISTING METHODOLOGY

Conceptual Model of Software Engineering Research approaches

Software has been a critical piece of current culture for quite a while. Specifically, this method is worried about various software development process models. Software process model is a portrayal of the sequence of activities completed in a software engineering project, and the relative request of these activities. It addresses a portion of the development models specifically, waterfall, v-shaped, incremental, RAD, iterative spiral and agile model.

Software engineering models from traditional method to modern technologies

A software lifecycle is the arrangement of recognizable stages that a software product goes through during its lifetime. However, an appropriately overseen project in a developed software engineering environment can reliably achieve this objective. This examination is stressed over the methodologies that assess the life cycle of software through the development models, which are known as software development life cycle. Thusly, we are addressing traditional for example Spiral models just as present day development methodologies like Agile methodologies that incorporates Extreme programming, Scrum, Feature Driven Development; Component based software development methodologies and so forth These models have advantages and disadvantages too.

Extreme Programming Method for Innovative Software Based on Systems Design

In software development, the waterfall model is usually utilized, particularly for huge scope software systems. For more limited size software development, agile software development approaches, for example, outrageous programming or scrums are utilized. Traditional software development methodologies are chiefly focused toward customer-driven development, and hence, new software methodologies are regularly not generally welcomed in the industry. We propose another software development methodology that is pointed toward developing innovative software utilizing computerized reasoning (AI), idea creation, value engineering, and systems design.

Continuous Delivery of Software on IoT Devices

Given the powerful environment and changing conditions on the Internet of Things (IoT), developers need to intermittently refresh software and convey new versions on brilliant devices and edge devices, for example, gateways. A software update can produce unanticipated vacations, or can likewise modify the device asset utilization. Subsequently, propose a methodology that manages the requirement for (semi)automating deployment, monitoring and visualization of the effect of software reports on devices activity. We use modeling to abstract the ideas that matter in the space of constant software delivery for IoT devices.

Model User Stories in Agile Software Development

Agile methodologies use client stories to catch software prerequisites. This regularly brings about colleagues over underscoring their comprehension of the objectives, without legitimate joining of objectives from different partners or Clients. Existing UML or other objective situated displaying strategies will in

general be excessively unpredictable for non-specialized partners to appropriately communicate their objectives and impart them to the agile group. In this paper, we propose a light weight Goal Net based strategy to demonstrate objective necessities in agile software advancement interaction to address this issue. Our starter investigation and studies in instructive software designing settings show that it can improve agile group's gathering attention to project objectives and, in this way, improve group productivity and artifact quality. The proposed approach was assessed in college level agile software designing ventures.

Search-Based Software Engineering

Highlight area is quite possibly the most significant and normal exercises performed by engineers during software maintenance and evolution. Highlights should be situated across groups of items and the software relics that understand each component should be recognized. In any case, when managing industrial software ancient rarities, the search space can be gigantic. a technique that estimates likenesses between literary questions. The algorithms are applied to two contextual analyses from our industrial partners (driving producers of home machines and moving stock) and are analyzed as far as exactness and review.

3. PROPOSED METHODOLOGY

Software comprehension represents more than 33% of the lifetime cost of a software system and the cycle of software comprehension is straightforwardly identified with complexity of software. The calculation of software complexity has been finished by the researchers using distinctive impacting credits like control flow paths, the volume of operands and administrators, identifier thickness, psychological complexity and spatial complexity. Spatial complexity measurements demonstrate the trouble of understanding the rationale of the program as far as lines of code that per user is needed to cross to follow control or data dependencies as they assemble a psychological model. Spatial complexity of article oriented software is the blend of class spatial complexity and item spatial complexity. The article spatial complexity depends on the meaning of objects and utilizations of item individuals. The classes don't straightforwardly execute typically, yet their examples are made in type of the objects in object-oriented software through which the usefulness of the classes is executed. The article spatial complexity appraises the spatial capacities expected to correspond different meanings of the objects with their particular classes, and different calls of methods to their separate definitions.

The least difficult of all software complexity estimations is the quantity of lines of code; the more prominent the quantity of lines, the more sophisticated a software system will be. Better estimation of complexity incorporates basic checks of program proclamations and investigation of a projects control structures.

Spatial complexity of object-oriented software class Spatial complexity (CSC), and object Spatial complexity (OCSC), which the class psychological spatial complexity (CSC) measures the Spatial complexity of the two individuals from the classes-strategies and qualities.

Spatial complexity metrics

Intelligence tests examine various cognitive abilities. Verbal ability is tested. Graphical and textual based tests are utilized to test induction, and spatial abilities are tested utilizing mental pivot assignments. Processing of location related visual data. Spatial ability has been connected with the selection of problem-tackling procedure and has assumed a significant part in the formulation of a powerful model of working memory.

Software Engineering Process Model

Software Processes is a rational arrangement of exercises for indicating, planning, carrying out and testing software systems.

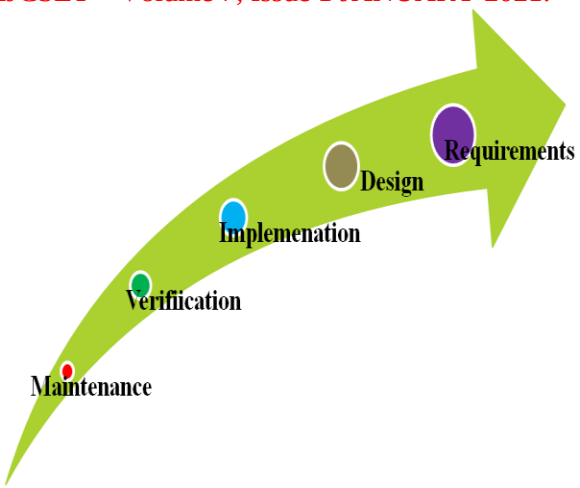

Figure 3.Software Process Model

A software life cycle model is either a descriptive or prescriptive characterization of how software is or ought to be created. A descriptive model depicts the historical backdrop of how a specific software framework was created. A prescriptive model endorses how another software framework ought to be created. Prescriptive models are utilized as rules or systems to put together and structure how software development activities ought to be performed, and in what request. This is conceivable since most such models are instinctive or all around contemplated. This implies that numerous quirky subtleties that depict how a software framework is an implicit practice can be disregarded, summed up, or conceded for later thought.

Spatial Complexity of Procedure-Oriented Software

Software comprehension is perhaps the biggest expense in the software lifecycle. While trying to control the expense of comprehension, different complexity metrics have been proposed to describe the trouble of understanding a program and along these lines permit precise assessment of the expense of a change. Spatial complexity metrics endeavor to represent the trouble of perusing the source code of a program for comprehension. The object spatial complexity measure can be utilized to gauge the comprehension of handling rationale through objects and their connection, which thusly reflects viable usage of the objects towards definite arrangement. Classes typically are not utilized straightforwardly, however through objects as it were. Lower estimation of object spatial complexity demonstrates that the class has been used through objects in closeness to the class affirmation, and, henceforth, understanding the use of that class towards complete software working will be much more straightforward than a class having a bigger estimation of article spatial intricacy.

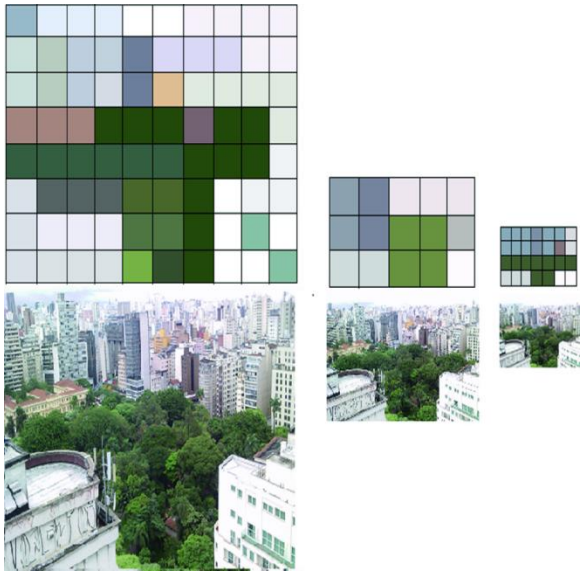

Figure 4. Spatial Complexity

The measures of cognitive complexity proposed by different creators thought about just these loads, which were impression of architectural perspective just and didn't investigate the spatial angle by any stretch of the imagination. Then again, the significance of spatial distance towards complexity is grounded and revealed. In this way it is exceptionally appropriate to consolidate the effect of architectural just as spatial parts of the software to register the cognitive complexity.

The program's code helps in understanding the handling rationale and the data factors and constants help in perceiving the input and output of the software. The spatial complexity dependent on the code is reliant on the definition and utilization of different segments of the software. In any case, there is numerous software, which handles loads of data and do similarly lesser preparing.

Cognitive Functional Size

The Cognitive Functional Size (CFS) metric proposed by Wang et al. basically measure algorithmic complexity of a program free of the language usage. The CFS metric can't demonstrate comprehension level of a program totally since the cognitive exertion needed to grasp a program likewise relies upon the programming language wherein the specific program has been composed as various languages have various levels and the language level has been accounted for to influence cognitive endeavors of understanding the programs. In this manner, the cognitive measures ought to likewise consider the usage subtleties of a program while estimating the exertion needed for comprehension of the program. Also, the spatial complexity measures depend on the spatial distance between the definition and utilization of different program components.

As of late, object-oriented metrics has been a region of expanding interest, not just from the agreement that data and procedure are united thus require the arrangement of new metrics, yet additionally from a practical perspective. Object-oriented dialects are getting progressively well known as a vehicle for the development of huge software frameworks.

Class Spatial Complexity

Measures of spatial complexity of object oriented programming are masterminded as class spatial complexity and object spatial complexity. These metrics are not simply the expansion of the spatial complexity metrics of procedure oriented software, yet these measures do deal with striking highlights of object oriented software. The understandability of the object oriented software begins with appreciating the idea of classes as an encapsulation of data and methods.

a. Class Attribute Spatial Complexity

The CASC of an attribute can be characterized as the normal of distances of different utilization of that attribute from its definition/past use.

$$\mathbf{CASC} = \sum_{\mathbf{I=1}}^{\mathbf{P}} \mathbf{Distance\ (I)/\ P} \text{ ----- (1)}$$

Where P addresses the check of utilization of that attribute and Distance (I) is equivalent to the outright contrast in number of lines of the current utilization of the attribute from its simply past use inside a similar method.

Distance = (distance of first use of the attribute from the top of the current file) + (distance of Definition of the attribute from the top of the file containing definition)-- (2)

Total class attribute spatial complexity of a class (TCASC) is characterized as normal of CASC, all things considered (factors just as constants) of that class.

$$\mathbf{TCASC} = \sum_{\mathbf{I=1}}^{\mathbf{q}} \mathbf{CASC(I)/\ q} \text{ ---------- (3)}$$

Where 'q' is the count of attributes in the class.

b. Class Method Spatial Complexity

The Class Method Spatial Complexity (CMSC) of a method is characterized as distance (in LOC) between the affirmation and the meaning of the method.

Distance = (distance of definition from the top of file containing definition) + (distance of declaration of the method from the top of the file containing declaration) ------- (4)

Total class method spatial complexity (TCMSC) of a class is characterized as normal of class method spatial complexity of all methods of the class.

$$\mathbf{TCMSC} = \sum_{\mathbf{I=1}}^{\mathbf{m}} \mathbf{CMSC(I)/\ m} \text{-------- (5)}$$

Where' m' is the check of the methods of the class The class is an encapsulation of attributes and methods, the class spatial complexity is a joining of the two sorts of spatial intricacies, and thus the class spatial complexity (CS C) of a class is proposed as,

$$\mathbf{CSC} = \mathbf{TCASC} + \mathbf{TCMSC} \text{---------- (6)}$$

Object-Oriented Spatial Complexity Metrics

The spatial complexity measures can be effectively altered to survey the complexity of object-oriented code, similarly as it tends to be adjusted to other literary programming dialects with no extraordinary level of trouble. There are two fundamental types of inheritance relations that are utilized inside object-oriented dialects, inheritance through class reuse and inheritance through the development of compound objects. A fourth measure, a composite measure, is additionally given.

Object definition spatial complexity

The object definition cognitive-spatial complexity (ODCSC) of an object is the result of the cognitive load of the BCS, wherein the object is being characterized and the total distinction (in LOC) of the definition of the object from its class revelation. Hence, Object Definition Cognitive-Spatial Complexity (ODCSC) of an object I at line number k is characterized as:

$$\mathbf{ODCS(i) = w_k^* Distance\ (i, k)}\text{-------- (7)}$$

Where Wk is the cognitive weight of the BCS, wherein the object, I has been characterized at line number k and Distance (I, k) is the outright contrast (in LOC) of the definition of the object from the comparing class assertion. In the event of numerous records, the distance is characterized as:

Distance = (distance of object definition from top of current file) + (distance of declaration of the corresponding class from the top of the file containing class) --- (8)

b. Object-Member Usage Spatial Complexity

A part through a specific object is characterized as the normal of distances (in LOC) between meanings of the part in the relating class and calls of that part through the object. Object Member Usage Cognitive-Spatial Complexity of an object part I at line number k is characterized as: -

$$\mathbf{OMUCSC(i, k) = W_k^* Distance(i, k)}\text{- ---- (9)}$$

Where Wk is the cognitive load of the BCS, wherein the object-part, I has been utilized at line number k and Distance (I, k) is the total distinction (in LOC) of the current utilization of the object-part from its definition in the relating class.

Distance = (distance of call from the top of the file containing call) + (distance of definition of the member from the top of the file containing definition)

Accordingly, Object Member Usage Cognitive-Spatial Complexity of an object-member I is characterized as the normal of cognitive spatial intricacies of all uses of the object-member i.e.

$$\mathbf{OMUCSC_i = \frac{\sum_{k=1}^{m} OMUCSC(i,k)}{m}}\text{---- (10)}$$

Input: A set of programs
Output: Calculated complexity of programs
For each input character
IF (input stream==for) THEN
Call class for ()
Complexity for-check ()
Index return index
IF (input stream==if) THEN
Call class IF ()
IF (input stream==SEQUENCE) THEN
Complexity SEQUENCE complexity
END FOR
PRINT complexity

Experiment Result

Compare the proposed measures with cognitive complexity measure (CFS) and spatial complexity measures (CSC and OSC) for programs showed Table 1.

	OO Cognitive Spatial Complexity Measures		Cognitive Complexity Measures	OO Spatial Complexity Measures	
	CCSC	OCSC	CFS	CSC	OSC
Java Program	12.49	114.62	12	6.57	62.48
C++ Program	12.78	120.41	12	6.58	64.57

Table 1.Computation Results for the Measures

Program	IBVCM	HSU	Proposed CF-OOSCM
Java	11.21	12.62	15.27
C++	12.43	15.41	16.57

Table 2.Comparison table of OO Cognitive Spatial Complexity Measures in CCSC

The table 2 shows the comparison table of OO Cognitive Spatial Complexity Measures in CCSC ratios demonstrates the existing algorithms IBVCM, HSU and proposed CF-OOSCM algorithm. The proposed algorithm is superior to the existing algorithm. The existing algorithms (IBVCM, HSU) values start from 11.21 to 12.43, 12.63 to 15.41 and, proposed Algorithm CF-OOSCM starts from 15.41 to 16.57provide the great results.

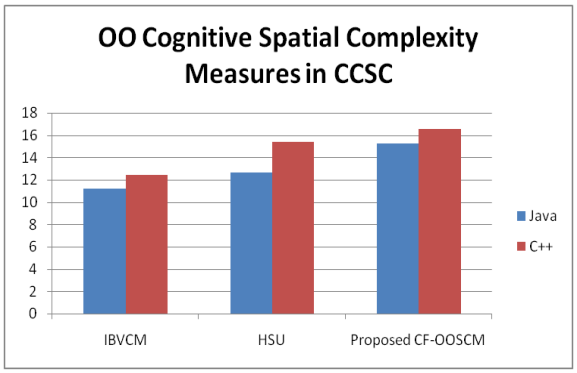

Figure 4.Comparison chart of OO Cognitive Spatial Complexity Measures in CCSC

The figure 4. shows the comparison chart of OO Cognitive Spatial Complexity Measures in CCSC ratios demonstrates the existing algorithms IBVCM, HSU and proposed CF-OOSCM algorithm. The proposed algorithm is superior to the existing algorithm. The existing algorithms (IBVCM, HSUvalues start from 11.21 to 12.43, 12.63 to 15.41 and, proposed Algorithm CF-OOSCM starts from 15.41 to 16.57provide the great results.

Program	IBVCM	HSU	Proposed CF-OOSCM
Java	12	14	16
C++	15	21	24

Table 3.Comparison table of Cognitive Complexity Measures in CFS

The table 3 shows the comparison table of OO Cognitive Spatial Complexity Measures in CFS ratios demonstrates the existing algorithms IBVCM, HSU and proposed CF-OOSCM algorithm. The proposed algorithm is superior to the existing algorithm. The existing algorithms (IBVCM, HSU) values start from 12 to 15, 14 to 21 and, proposed Algorithm CF-OOSCM starts from 16 to 24 provide the great results.

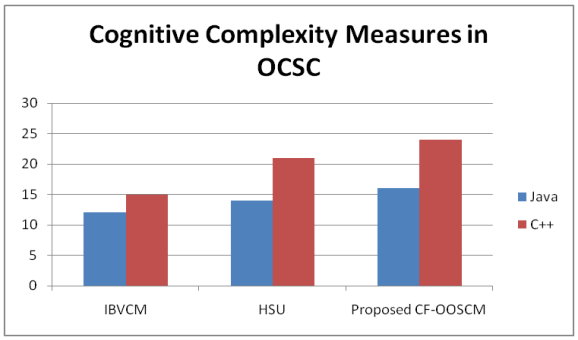

Figure 5.Comparison chart of OO Cognitive Complexity Measures in CFS

The figure 5 shows the comparison chart of OO Cognitive Complexity Measures in CFS ratios demonstrates the existing algorithms IBVCM, HSU and proposed CF-OOSCM algorithm. The proposed algorithm is superior to the existing algorithm. The existing algorithms (IBVCM, HSU) values start from values start from 12 to 15, 14 to 21 and, proposed Algorithm CF-OOSCM starts from 16 to 24 provide the great results.

CONCLUSION

Spatial capacity is a term that is utilized to allude to a person's cognitive abilities identifying with orientation, the location of objects in space, and the handling of location related visual data. To effectively address debugging, maintenance and comprehension tasks, programmers should have information on the programming language, have a comprehension of the application domain and build up an appreciation of the connections that can exist between the two. Program comprehension and software maintenance are considered to considerably utilize programmers spatial abilities.

In this paper, creator has proposed new cognitive-spatial complexity measures for object-oriented software. The proposed measures accept spatial just as architectural complexity of the software into represent the assessment of the cognitive exertion needed for software comprehension measure. This relative investigation has shown that the proposed measures are better pointers of the cognitive exertion needed for program comprehension than the comparing existing cognitive and spatial complexity measures.

In future focusing on behavioral between moms housed in one or the other fundamental or enhanced cages, we identified just an impact of cage type in feeding and drinking conduct. We utilized a thorough and set up echogram covering every key conduct (maternal and non-maternal), accordingly, it is improbable we missed any practices for which females in the two cage types may vary. Additionally looking at traditional enrichment effects and upgraded spatial complexity as executing in two layer confining frameworks examination of effects on task performance as traditional enrichment.

REFERENCE

[1]. A. Furfaro, T. Gallo, A. Garro, D. Saccà and A. Tundis, "ResDevOps: A Software Engineering Framework for Achieving Long-Lasting Complex Systems," 2016 IEEE 24th International Requirements Engineering Conference (RE), Beijing, 2016, pp. 246-255, doi: 10.1109/RE.2016.15,IEEE.

[2]. Chengying Mao, Changfu Xu, “Entropy Based Dynamic Complexity Metrics for Service Oriented Systems”, 24th Asia-Pacific Software Engineering Conference Workshops, IEEE, 2017.

[3]. Dr. Rakesh Kumar, Gurvinder Kaur, “Comparing Complexity in Accordance with Object Oriented Metrics”, International Journal of Computer Applications, Volume 15, Issue 8, February 2011, PP42-45.

[4]. G. Kumar and P. K. Bhatia, "Comparative Analysis of Software Engineering Models from Traditional to Modern Methodologies," 2014 Fourth International Conference on Advanced Computing & Communication Technologies, Rohtak, 2014, pp. 189-196, doi: 10.1109/ACCT.2014.73.IEEE

[5]. Gayatri Vijiyan 2014 “Current Trends in Software Engineering Research”DOI: 10.18488/journal.76/2015.2.3/76.3.65.70.IEEE.

[6]. IqbalH.Sarker Md., Faisal Faruque 2015 “A Conceptual Model of Software Engineering Research approaches”, pp. 229-236, Doi: 10.1109/ASWEC.2009.42. IEEE.

[7]. J. K. Chhabra, K. K. Aggarwal, and Y. Singh, “A unified measure of complexity of object-oriented software”, Journal of the Computer Society of India, vol. 34(3), 2004, pp. 2-13.

[8]. J. Lin, H. Yu, Z. Shen and C. Miao, "Using goal net to model user stories in agile software development," 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Las Vegas, NV, 2014, pp. 1-6, doi: 10.1109/SNPD.2014.6888731,IEEE.

[9]. M. H. Sadi and E. Yu, "Analyzing the evolution of software development: From creative chaos to software ecosystems," 2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS), Marrakech, 2014, pp. 1-11, doi: 10.1109/RCIS.2014.6861055,IEEE.

[10]. M. Kassab, J. DeFranco and V. Graciano Neto, "An Empirical Investigation on the Satisfaction Levels with the Requirements Engineering Practices: Agile vs. Waterfall," 2018 IEEE International Professional Communication Conference (ProComm), Toronto, ON, 2018, pp. 118-124, doi: 10.1109/ProComm.2018.00033,IEEE.

[11]. P. Singh, H. Singh, “Class-level Dynamic Coupling Metrics for Static and Dynamic Analysis of Object-oriented Systems”, International Journal of Information and Telecommunication Technology, 1(1): 16-28, 2010.

[12]. S. Misra and A. K. Misra, “Evaluation and comparison of cognitive complexity measure”, ACM SIGSOFT Software Engineering Notes, vol. 32(2), 2007, pp. 1-5.

[13]. Shweta Sharma, Dr. S. Srinivasan, “A review of Coupling and Cohesion metrics in Object Oriented Environment”, International Journal of Computer Science & Engineering Technology (IJCSET), ISSN: 2229-3345, Vol. 4 No. 08, Aug 2013.
[14]. T. Klemola and J. Rilling, “A Cognitive Complexity Metric Based on Category Learning”, Proc.Second Int’l Conf. Cognitive Informatics, pp. 106-112, 2003.
[15]. Y. Chen, "Modeling Information Security Threats for Smart Grid Applications by Using Software Engineering and Risk Management," 2018 IEEE International Conference on Smart Energy Grid Engineering (SEGE), Oshawa, ON, 2018, pp. 128-132, doi: 10.1109/SEGE.2018.8499431,IEEE.