IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-7, Issue-3, March 2019 E-ISSN: 2347-2693

Designing The Code Snippets for Experiments on Code Comprehension of Different Software Constructs

Leena Jain[1*], Satinderjit Singh[2]

[1]Department of Computer Applications, GIMET, Amritsar, PTU Kapurthala, India
[2]Department of Computer Applications, GGNIMT, Ludhiana, PTU Kapurthala, India

[*]*Corresponding Author: satty74@gmail.com, Tel.: +91 9814736494*

DOI: https://doi.org/10.26438/ijcse/v7i3.310318 | Available online at: www.ijcseonline.org

Accepted: 23/Mar/2019, Published: 31/Mar/2019

***Abstract*—**The concept of Basic control structure (BCS) of software and their cognitive weights have been proposed in theory. However not much work has been done to validate weights assigned to various programming constructs. One of the primary reasons for same is that it is difficult to design the experiment to measure the mental effort involved in understanding the effect of various programming constructs and their interplay. The paper discusses some of the challenge involved in setting up such psychological experiment. In such experiments we cannot select and compare any random code snippets of various programming constructs- the variations are endless. We identified different approaches to conduct such experiments. We explained with example various factors and issues involved in selecting the code snippets which resulted in minimum variations in code snippets of various programming constructs, other than that is inherent in syntax. The code snippets design approach proposed here can be used to conduct series of psychological experiments in software studies. We need series of such experiments not only to validate the cognitive weights of different programming constructs, but also it will go long way in having robust metrics for software complexities. These types of experiments can be extremely useful in the field of computer science education in understanding the cognitive load required for learning the concepts of programming languages.

***Keywords*—** *Software Complexity;Code complexity;Code Comprehension; Cognitive Weights;Basic control structure;cognitive metrics;Cognitive load; Software Experimentation; computer science education; Code snippets;human brain working.*

I. INTRODUCTION

Many measures like LOC, Halstead measures, McCabe's cyclometric [1] measures were proposed earlier to measure the complexity hidden inside the software. All these measures capture some aspect of complexity while ignoring the others. However, none of these satisfactorily captures the human aspect of software complexity i.e. mental effort required to understand the software code. In year 2003 Shao and wang proposed the measure cognitive weights of 10 BCS and proposed new measure cognitive functional size to express the complexity of the software [2]. In 2006 Wang modified the cognitive weights of the 10 BCS after the series of psychological experiments conducted on students [3], [4]. Post that slew of metrices were proposed by various researcher based on the cognitive weights proposed in 2003 and 2006 [5], [6]–[9].

Gruhn and Laue in the year 2007 pointed various flaws in the psychological experiments conducted by wang [10]. He also highlighted some of the issues in measuring the cognitive weights of recursion BCS owing to its peculiar nature. He also suggested that we must include three more control structure in the set of BCS – named as lock, exception and internal exits – into the table set of BCS. He highlighted some of the precautions which should be covered in designing the any such experiments. In an unpublished work (But available in Research Gate website) David Admino in 2015 conducted the same sort of experiments and achieved quite a different result than wang's result [11]. There are other works where researchers have compared various control structure and their effects in code complexities [12]–[17]. Interesting work is done in paper by Ajmi etal 2017; in which they have conducted the experiment and shown that complexity of code is not only dependent upon control structure but also on factors like different ways to express the predicate and different idioms used (in say looping structure) [18]. In original wang and Shao metrics in 2003, 2006 the latter factors were not considered in complexity calculation of software code [2], [3]. Most of complexity metrics based upon wangs work and even non-cognitive complexity measures like has also not considered the latter factors [5], [7]–[9], [19], [20]. So, there is urgent need to not only validate the cognitive weights proposed by wang and others

© 2019, IJCSE All Rights Reserved

but also to see if some other factors affect the software complexity. It is surprising that apart from some works mentioned above there is not much body of work done – in terms of experimentation, explanatory theory in Brain science- to validates the weights assigned to the BCS.

In this paper we discuss some of the issues involved in setting up the experimentation to measure various BCS weights. We look at the aspect of how to design the code snippets for conducting the psychological experiments to measure BCS. We suggest some of the approaches to design as homogeneous code as possible where only the intrinsic difference in BCS are measured and all other differences are cancelled out as far as possible. We also presented the sample code snippets pertaining to some BCS along with all the necessary parameters to clarify our points. We also discussed the possibility of future scope of this kind of work and the direction it should take.

The paper is divided into 7 section. In section I we provide the general background of the topic. Section II provides the previous cognitive weights and their values in comparative forms. Section III discuss the experimental works done in this regards and issues in them. In section IV we discuss the issues and challenges involved in design of the psychological experiment to measure the weights of BCS of software. In section V we specifically concentrate on the issues of design of code snippet set to conduct these sorts of experiments. In section VI we further explain one approach to design code snippets with the help of two examples and analyzing them. In section VII we conclude by summarizing the discussion done in previous sections.

II. COGNITIVE WEIGHTS OF BCS

In the year 2003 Yingxu Wang introduced the concept of cognitive functional complexity of software [2]. In this metrics the BCS basic control structures are assigned cognitive weights. BCS are the set of fundamental and essential flow control mechanisms that are used for building logical architecture of software.

In this metrics the total cognitive weight of a component is measured by either adding the weights of a BCS if they are in series or they are multiplied if they are embedded in another BCS. The total cognitive weight of a software component, Wc is defined as the sum of cognitive weights of its q linear blocks composed in individual BCS's. Since each block may consist of m layers of nesting BCS's, and each layer with n linear BCS's, the total cognitive weight, Wc can be calculated by equation (1).

$$Wc = \sum_{j=1}^{q}\left[\prod_{k=1}^{m}\sum_{I=1}^{n}\left(Wc(j,k,i)\right)\right] \qquad (1)$$

In this metrics the different BCS are assigned the weights as shown in table 1. These weights are based on the human effort in comprehending these BCS.

The cognitive functional size (CFS) of a basic software component that only consists of one method, S_f, is defined as a product of the sum of inputs and outputs ($N_{i/o}$), and the total cognitive weight, i.e.:

$$S_f = N_{i/o} * W_C . \qquad (2)$$

However, in the year 2006, Wang [3] suggested new weights for various BCS as mentioned in Table 2.

Although the weights for BCS were changed by Wang, but the method of calculating overall cognitive complexity of the software remains same. In an unpublished work (But available in Research Gate website) David Admino in 2015 conducted the same sort of experiments and achieved quite a different result than wang's [11]. These are shown in Table 3.

Gruhn and Laue (2007) suggested that we should add three more BCS other than 10 mentioned in tables above[10]. The new BCS identified and named by them are *lock*, *exception* and *internal exits*. There is large body of work done by host of researchers in refinement of how to calculate complexity metrics from code [6], [8], [9]. There are also a decent set of researchers who have proposed complexity metrics in which they have recognized the concept of BCS but have used their own weights and complexity calculation method other than as proposed by Wang and shao [19], [20]. Clearly the well accepted cognitive weights of BCS are critical issues in here and without their validation there will always be question mark on acceptance aspect of various cognition-based complexity metrics. Apart from this there are still some troubling questions about Cognitive Metrics which we discuss in our next section.

III. ISSUES RELATED WITH COGNITIVE METRICS

The objection of Gurhn and Laue [10] and different method to calculate Complexity metrics apart, there are some other key concerns regarding Software cognitive Metrics. Some of the concerns and doubts regarding BCS values are put by Jain and Singh [21]

1) How authentic the Cognitive weights of BCS are? Can we verify them experimentally?
2) Do these cognitive weights fit all population?
3) Is it possible to identify various segments of the population which does not concur with above weights of BCS but have their own set of cognitive weights?

© 2019, IJCSE All Rights Reserved

These and many more questions and doubts of the same category are extremely valid and important for the researcher community.

IV. PREVIOUS WORK IN COGNITIVE WEIGHTS

To the best of our knowledge there are three works which has been carried out to measure cognitive weights of BCS. The first one is obviously by wang and shao in 2003, when they proposed the concept of BCS and their weights [2]. The second one is when in 2006 wang again published a work where they modified BCS weights with new values [3]. Then there is unpublished work of David Adamo Jr (but available in Research gate website) in 2014 [11], in which he works out an experiment and reach value quite different from above two values.

Table 1: Cognitive Weights of different BCS-Wang 2003.

Category	BCS	Cognitive weights (W_c)
Sequence	*Sequence*	1
Branch	*If then else*	2
	Case	3
Iteration	*For-loop*	3
	Repeat-loop	3
	While-loop	3
Embedded Component	*Function call*	2
	Recursion	3
Concurrency	*Parallel*	4
	Interrupt	4

The first work of wang and shao [2] do comes out with weights and claims that the weights have been calculated as the direct relationship to the time consume in working out output in the code snippets of various BCS. But the paper does not explain the details of the experiment conducted. So Gurhn and Laue (2007) correctly stated that academic value of that work cannot be taken seriously [10].

In second paper wang [3] do provide the detail of the experiment layout. In an experiment consisting of 122 undergraduate and post graduate students, certain code snippets were given to them in Java language. The students' response time were measured and based on the time calculated for each BCS, the cognitive weights are calculated by dividing the BCS time with the sequential BCS time. Gurhn and Laue (2007) pointed out three faults in experimental layout [10]. First that there is syntax problem with the java code snippets. Second that there is too much variance in code length (in terms of LOC) of various code snippets pertaining to various BCS. The argument is that since we are only measuring time and code must be read before interpreted, so it only fair that code length should be almost same if not exactly same. In the paper of wang [3] the two snippets had 7 and 22 LOC, thereby rendering the result quite questionable. The third fault pointed out by Gurhn and Laue is that wang had not used the correctness feature of the result. In-fact paper does not provide any information about the correctness percentage of the answer. Basili and others had suggested this as important indicator in the code comprehending experiments[12], [22], [23].

Table 2: Cognitive Weights of different BCS- Wang 2006

Category	BCS	Cognitive weights (W_c)
Sequence	*Sequence*	1
Branch	*If then else*	3
	Case	4
Iteration	*For-loop*	7
	Repeat-loop	7
	While-loop	8
Embedded Component	*Function call*	7
	Recursion	11
Concurrency	*Parallel*	15
	Interrupt	22

Third work in this area is done by David Adamo Jr [11]. This work does take into consideration the limitations of wang work [3] and tries to incorporate the suggestion by Gurhn and Laue [10]. It tries to eliminate the difference due to varying LOC of code snippets. However, two limitation of David

Adamo [11] cannot be ignored. the first is that it works out BCS weight value of only 9 BCS – excluding interrupt BCS. The second more important limitation is that experiment is conducted on only 14 undergraduate students. The number we feel is too small especially considering the result it arrived at -variant both in values but also in rank order of various BCS. We also believe that one aspect is missed by all the previous work in this field. In the choice of the code snippets the mathematical and logical operators are too varying. Since we are using time and correctness parameters to work out cognitive values, we believe that there must be homogenization in operator numbers and type within the

© 2019, IJCSE All Rights Reserved

various code snippets as far as possible. This should cancel out the effect of complexity arising out of operator variance. None of the previous work had taken this into consideration. More about this will be discussed in coming sections.

Table 3: Cognitive Weights of different BCS -David Admino 2015

Category	BCS	Cognitive weights (W_c)
Sequence	*Sequence*	1
Branch	*If then else*	2
	Case	2
Iteration	*For-loop*	11
	Repeat-loop	10
	While-loop	6
Embedded Component	*Function call*	Not Calculated
	Recursion	7
Concurrency	*Parallel*	Not Calculated
	Interrupt	Not Calculated

In addition to these works there has been other experiments done not based on BCS but based on some specific aspect of the code. Mynatt [14] conducted experiments to measure which one is psychologically more complex -Recursion or iteration- or which data structure is more complicated - Arrays or link list. Iselin [15] conducted an experiment to study the effect of positive/negative condition and its interaction with true /false condition. Ajami etal [18] conducted experiment to show that 'for' loop is complex to comprehend than 'if' statement. Also, other findings are that some logical negation does affect the predicate in more complex way than otherwise and counting-down in loop is harder than counting-up. However, the feature of these types of experiments is that no weights are developed; only qualitative assessment is done of complexity hidden on certain aspect of code.

Amount of the work done in this field of software engineering is surprisingly less. One reason is that it is difficult to set up the experimentation where only thing measured is corresponds to complexity inherent in BCS structure and not any other aspect of the code. The variation in minutest of the code snippet can be endless. But this point also must be noted that we must conduct multiple of these psychological experiments before we go on to develop better complexity metrices or validate the values of BCS if there exists any unique one. In next section we talk about the challenges and issues in designing the layout for conducting psychological experiments of these kinds.

V. GUIDELINES FOR THE LAYOUT OF THE EXPERIMENTS

To the researcher community – both in the field of software and otherwise- validating the BCS cognitive weights is of outmost importance. One of the reasons that too little work has been done on this side is the difficulty in conducting the Psychological experiment of this kind on the software code. Ajami etal [18] has suggested two problem with the experimental setup. The first is the fact that according to them there are too many variables and factors hidden in code snippets that to single out each is tedious job. Second, they opined that considerable complexity of code lies out of the bounds of BCS as in composition of condition statement, use of programming idioms etc. Gurhn and Laue [10] have given following guidelines for conducting the experiments of these types. These include

- Non-variance of code length, variables-Number, types and names.
- Multi-language experiments.
- Different level of respondents in terms of experience.
- Enough replications.
- Following the strict approach of experimental study.

In the remaining portion of this section we suggest in similar sense some new points and some possible refinement of the above said points.

1. On the issue with idea of same code length for all snippets to be compared, we believe that strict adherence to that may not be that prudent thing. Firstly, because different language may show different LOC of code length for same logic expression. Secondly, it is in the inherent structure of various BCS that LOC variations are there. For example, in comparable code snippets of sequential and if-else BCS LOC will be different because if-else will have at least one condition statement and possible pair of curly braces; thereby increasing the LOC of later. Ajami etal [18] has shown that in comparison to single logical expression in condition statements, the same logic when implemented through nested if-else takes less time to solve than latter, although nested if-else has lot many higher LOC than single compound logical expression. It's not that we are suggesting complete abdication of this code length factor. All we are suggesting that this cannot be binding principles for conducting such experiments, yet we continue to hold that LOC of code snippets should not vary widely.

© 2019, IJCSE All Rights Reserved

2. In the same spirit the number and types of operators can also vary across the BCS. In case of looping and control BCS, it is imperative that we have in conditional statement relational/ logical operators as a part of code structure of these BCS. In fact, not having these relational /logical operators in looping and conditional BCS would enhance the code complexity of these BCS most likely.

3. What should be non-variant in these experiments involving comparing code snippets results is that number of mathematical operators which the respondents used to calculate output from given input. The number and types of variables (int, float, char etc) and operators should be same. By doing that we tend to eliminate the variation which result in calculation of different operators.

4. Since code comprehension experiments involved measuring the timing and accuracy parameters for each BCS, care should be taken of proper coverage of BCS code. This can be done for example by giving a set of input that all nested part of if-else or all cases of switch statement including the default case are executed and their time and accuracy is measured.

5. Following the same principal as stated above in case of looping structure, the experiments should involve loop portion in code snippets to run multiple number of times- including minimum number of times (0 or 1). It is further suggested that in comparing looping structures – for, while, do-while and even recursion- all the corresponding code snippets must have the loop run for same number of times just to cancel out the effect of different number of runs.

6. The variation in code snippets as the control factor is at the heart of such Psychological experiments. This is true whether we are measuring the BCS effect or even other features of coding. Ajami etal [18] has suggested that variations in simplest of code are endless, and many factors are at play, so meaningful experimental design may be tricky in such experiments. Although there are many permutations and combination possible, we suggest that following broad snippet design approaches should be considered.

    a) Code snippets with same number of LOC, operators and variables.
    b) Code snippets with same number of variables and non-inherent operators.
    c) Code snippets with same variables and same number of effective non-inherent operators.
    d) Code snippets with same logic only.

In an ideal world the design approach (a) would be perfect. Not only it will adhere to guidelines of same LOC as suggested by Gurhn and Laue [10] but it imposes another condition of same number and types of operators and variables. By doing this we eliminate the variation which can come about due to varying LOC and due to different operators and different varying variables type. Since we believe – based on experience- that not all operators are same, and they do have varying complexity. It would be interesting for future research work if someone works out the order or ratio of the complexities associated with various operators. But for now, we have tried to eliminate that variation by suggesting code snippets with same set of operators and variables. However, one possible limitation of such design scheme is that in the rigidity of design we may miss the essence of the BCS itself -whose cognitive weights we intend to measure. For example, it is natural to have at least one relational operator in if-else structure. Now if the design scheme (a) is to be followed then either we must introduce relational operator in Sequence BCS or we must remove the relational operator from condition part of 'if-else' structure. In both the case we are distorting the essence of either of BCS. Another issue that pertains to design (a) is same number of LOC of various code snippets in design may not be constant across languages. This sameness may vanish if we translate code snippet into some other language.

The rigidity inherent in scheme (a) made us suggest design approach b). Wangs [3] and later David Adamo [11] experiment was partially (not fully) based upon this approach. Although we agree with Gurhn and Laue [10] objection of too much variation between LOC of code snippets. The idea here is not to be too rigid with code length or operators used, but in same breadth not to lose overall sight of these two important factors and allowing variation only when needed to retain the essence of BCS differences. By non-inherent operators we mean those operators which are not part of BCS itself. For example, in 'while' BCS at least one conditional operator is inherent in structure of BCS. So, it will not be counted, and remaining operators became the non-inherent operators.

The fourth approach of having a code snippet set with same problem but applied with different logic suited to various BCS. Ajami etal [18] followed this design approach in working out various parameters involved in code complexities. The only problem it seems is that it may be tricky to find the problem and different logics to solve it. Yet all BCS may not be captured in various code snippets generated through this design approach.

© 2019, IJCSE All Rights Reserved

The third approach to design code snippets is novel one. To the best of our knowledge this has never been implemented before to conduct any experiments of this sort. The simple philosophy here is that since we are measuring the respondents of code snippets in terms of time and correctness measurement in psychological experiments, so we must ensure that mathematical/logical operators must be same in terms of type and numbers. For example, a switch BCS having three cases – two simple and one default case- must have for all three cases same operation on same type of variables. Of course, the order of operator usage and variable value should differ to avoid sameness effects in various code snippets. By this we may have code length roughly three times that of sequence BCS. Ajami etal [18] has already shown that LOC is not a strict indicator of code complexity in certain cases. Also, we propose that if overall all the code snippet is of short size (Less than 20 LOC) then this variation of LOC in various code snippets may not be that crucial factor.

The very nature BCS is such that it suggests selecting the specific part of code for further processing – both in computer and in human mind. Our endeavour here is to keep that processing part same and assume that what is left out to calculate is the inherent complexity in the structure of BCS. In the following section we show some code snippets designed using this approach.

VI. CODE SNIPPETS EXAMPLES

In this section we will give example of code snippet to be designed by this new approach -category c) of previous section. Gurhn and Laue [10] highlighted the point the that LOC of various code snippet should be of same size, but they missed the point that in psychological experiments of such kind-where we are measuring time and accuracy- the number of mathematical/logical operators are more important than LOC factor. Also, the LOC of code snippet may vary depending upon language to language. In this new approach we gave more importance to mathematical/logical operators encountered in a processing of input to output than LOC of code snippets. We believe that with small code snippets (less than 20 LOC) the variation in size can be ignored in favour of experimental accuracy and retaining the true essence of various BCS.

Example 1: Below in table 4 we gave an example of code snippets for four BCS – 'while', 'sequential', 'for', 'function'- that follows new design approach. The code is in C language. The table also provides information lines of code (LOC) and character strength (without space). Table 5 provides the operator analysis for the code snippets in table 4. We are interested in number of variables, number of operators in code snippet, number of operator inherent in BCS, number of variable used in mathematical/logical processing output from input (excluding the one which is inherent in BCS) and number of mathematical / logical operator encountered in processing-out output - both for human and computer.

Some points to clarify the information in table 4 and table 5 is as given below:

- The code snippets vary in terms of size; both in terms of LOC or character strength (without space). For an obvious reason the smallest code snippet is 1.2 – Sequential BCS- and the largest code snippet is 1.3 – 'for' BCS, although 1.4 – 'function' BCS- matches 1.3 in character strength but not on LOC. It is in the very nature of various BCS -syntax and design- that lack of similarity in code snippet size is almost inherent. And, if we try to make code snippets of same size then we may end up undermining the essence of some BCS.

Table 4: Code Snippet – Example 1.

S.No.	BCS	Code snippet	Var	LOC	Char (ws)

1.1	***While***	int a=9, b =4, c=0;
		while(a>1)
		{
		c=c+b*2;
		a=a-3;
		}
		printf("\n%d" , c);
			3
			{a,b,c}
				7	58

1.2	***sequential***	int a= 9, b =4,c;
		b=b-2;
		c= a+b*3;
		printf('\n%d",c);
			3
			{a,b,c}
				4	44

1.3	***if-else***	int a=5,b=6, c;
		if (a>b)
		{
		b= b -a;
		c=a+b*7;
		}
		else
		{
		a=b*a+7;
		c=a-b;
		}
		printf("\n %d", c);
			3
			{a,b,c}
				12	73

1.4	***Function***	int funct (int a,int b)
		{
		b= 3*b+2;
		return (a-b)
		}
		int c;
		c= funct(7,2);
		printf('\n %d" , c);
			3
			{a,b,c}
				8	75

Table 5: Operator analysis of Code snippets (Table 4)

© 2019, IJCSE All Rights Reserved

	Mathematical/ logical Operators	*Operators Inherent in BCS*	*Non-Inherent operators*	*Operators encountered in execution*
while	4 {+ , - , * , >)	1 { < }	3 { +, -, * }	3 { +, -, * }
sequential	3 {+ , - , * }	nil	3 { +, -, * }	3 { +, -, * }
if-else	7 { +(2) , -(2), *(2), > }	1 { < }	6 { +(2) , -(2), *(2) }	3 { +, -, * }
function	3 {+ , - , * }	nil	3 { +, -, * }	3 { +, -, * }

- Another small point to mention here is that we have deliberately picked the value of variable 'a' and 'b' as single digit number. All operation in code snippets have single digit operands. We have deliberately avoided multiplication with 10. All this is done to make mathematical operations simple and basic. Similar precautions need to be taken in case of division and remainder operations.

- The number of variables used in all code snippets of Table 4 is three. It is important that we use the same numbers and types of variables (int, float etc.) in various code snippets to control any result deviation coming out of the variation of the variables. It's not necessary that we should use same variables name (although we have here), but it is suggested that all variables names are neutral and meaningless. Esther et al [5] has used the different variable weight of meaningful named variable (MNV) than arbitrarily named variable (ANV) in overall complexity calculations.

- The crucial part is number of logical/mathematical operators in various code snippets. As the table 5 shows, the number of mathematical/logical operators in the code snippets vary for each BCS. Most of code snippets vary in number of the operator it used. It is important to note that we are only considering mathematical/ logical operators and not say assignment or comma operator. The idea here is to cover the mathematical or logical part of code processing only.

- In Table 5, we also make provision of operator that are inherent in BCS of code snippet – code 1.1 and 1.3. The idea is that these operators are inherent in BCS and are not used in output calculation but rather in selecting the code part to be executed. So, we eliminate that in our effective number of operators' calculation. This single adjustment allows us to have different types and numbers operators in various BCS code snippet -as is inherent in their structure- yet can be compared.

- The last column in Table 5 provides actual number of mathematical and logical operators encountered in path of the code in calculating output from input. Again, we are not counting same operators when a loop runs through portion of code again and again. The idea again is that this factor is inherent in BCS structure of loop and thus the calculation of operators should not be repeated. The last column shows the constant value of effective non-inherent operators used- which is whole purpose of code snippet design.

Table 6: Code Snippet – Example 2

S.No.	BCS	Code snippet	Var	LOC	Char (ws)

2.1	do-while	int m=87, n=7,p=0;
		do
		{
		m=m/n;
		p=p*10 +m%n;
		} while(m>0);
		printf ("\n %d", p);
			3 {m,n,p}	7	66

2.2	for	int m, n=6,p=0;
		for{
		m=53;m>0;m=m/n)
		{
		m=m%n;
		p=(p+m)*10 ;
		}
		printf ("\n %d", p);
			3 {m,n,p}	7	65

Table 7: Operator analysis of Code snippets (Table 6)

BCS	*Mathematical/ logical Operators*	*operators inherent in BCS*	*Balance Non-inherent operators*	*Operators encountered in execution*
while	5 {+ , / , * , % , >)	1 {>}	4 { +, / , * , %}	4 { +, / , * , %}
for	5 {+ , / , * , % , >)	1 { >}	4 { +, / , * , %}	4 { +, / , * , %}

Example 2: In example 2 of code snippet design we provide a slightly more complex code snippet for two BCS – "for" and "do-while" BCS in Table 6 and Table 7. Notice that there are just 3 variables in code snippets -just like example 1- however one variable is assigned two-digit number, and another single digit. Number of non-inherent operators works out to be 4 (as compared to 3 in example 1). Also, note that that in both snippets there is one multiplier by 10. We assume multiplier by 10 as easier to calculate than any other number, so either include in all the code snippets or exclude from all snippets (In this case we include it in both snippets). It is interesting to note that LOC and character strength of 2.1 and 2.1 are less than 1.3 and 1.4. But they have larger number and different types of operators in calculations

© 2019, IJCSE All Rights Reserved

needed to come out with output from input. These factors along with BCS inherent complexity may be the reason which may enhances the complexity for 2.1 and 2.2 than 1.3 and 1.4.

VII. FUTURE SCOPE AND CONCLUSION.

There is very little work done in terms of conducting the psychological experiments to measure the code complexity of the software. One reason for that could be difficulty in designing the code snippets so that we can measure the different aspect of the code including the control structure and its effect on complexity. Four-way classification of code snippet design approach has been explained in paper including the novel approach of equal effective non-inherent operators. Two small examples have been given to demonstrate how code snippets are designed and what factors need to be taken care for proper design under the novel approach.

Going further we suggest that large body of code snippets in various language are designed at various complexity level (number of operators, variables etc) so that body of work can be used by future researchers to conduct various experiments and many entangled questions -pertaining to cognitive weights of BCS- can be answered. We also believe that such work of generating code snippets can perhaps be automated in near future. This will go long way in developing and validating the truly acceptable software complexity measure.
One reason why we have not been able to develop generally accepted software complexity measure or not validate the cognitive weights of BCS is because we know so little of how our brains work – How they process, retain and use internal and external information. Software, perhaps more than any other field, involves this understanding of human brain. By conducting such experiments on software code snippets or preparing a layout for any such experiments, we may well be helping in understanding the working of human brain. This may benefit us not only with better software complexity measure but in many other ways. it can be useful to mankind in generating a better Brain understanding in general but more specifically this can also help us make better AI machine.

REFERENCES

[1] T. J. McCabe, "*A Complexity Measure*," IEEE Trans. Softw. Eng., vol. SE-2, no. 4, pp. 308–320, 1976.

[2] J. Shao and Y. Wang, "*A new measure of software complexity based on cognitive weights*," *Can. J. Elect. Comput. Eng.*, vol. 28, no. 2, pp. 1–6, 2003.

[3] Y. Wang, "*Cognitive Complexity of Software and its Measurement*," in 5th IEEE International Conference on Cognitive Informatics, 2006, pp. 226–235.

[4] Y. Wang and S. Patel, "IJSSCI-1201-CogFundSE.pdf," *Int. J. Softw. Sci. Comput. Intell.*, vol. 1, no. June, pp. 1–19, 2009.

[5] O. Esther, O. Stephen, O. Elijah, A. Rafiu, T. Dimple, and Y. Olajide, "*Development of an Improved Cognitive Complexity Metrics for Object- Oriented Codes*," Br. J. Math. Comput. Sci., vol. 18, no. 2, pp. 1–11, Jan. 2016.

[6] A. K. Jakhar and K. Rajnish, "*A new cognitive approach to measure the complexity of software's,*" Int. J. Softw. Eng. its Appl., vol. 8, no. 7, pp. 185–198, 2014.

[7] D. De Silval and N. Kodagoda, "*Improvements to a Complexity Metric : CB Measure,"* in IEEE 10th International Conference on Industrial and Information Systems, ICIIS 2015, 2015, pp. 401–406.

[8] S. Misra, A. Adewumi, L. Fernandez-Sanz, and R. Damasevicius, *"A Suite of Object Oriented Cognitive Complexity Metrics*," IEEE Access, vol. 6, no. January, pp. 8782–8796, 2018.

[9] S. Misra, A. Adewumi, R. Damasevicius, and R. Maskeliunas, "*Analysis of Existing Software Cognitive Complexity Measures*," Int. J. Secur. Softw. Eng., vol. 8, no. 4, pp. 51–71, Oct. 2017.

[10] V. Gruhn and R. Laue, "*On Experiments for Measuring Cognitive Weights for Software Control Structures*," in 6th IEEE International Conference on Cognitive Informatics, 2007, no. September 2007, pp. 116–119.

[11] D. J. Adamo, "*An Experiment to Measure the Cognitive Weights of Code Control Structures*." pp. 1–16, Jul-2014.

[12] M. E. Hansen, A. Lumsdaine, and R. L. Goldstone, "*An experiment on the cognitive complexity of code*," in Proceedings of the Thirty-Fifth Annual Conference of the Cognitive Science Society, 2013.

[13] H. Sackman, W. J. Erikson, and E. E. Grant, "*Exploratory experimental studies comparing online and offline programming performance*," Communications of the ACM, vol. 11, no. 1, pp. 3–11, Jan-1968.

[14] B. T. Mynatt, "*The effect of semantic complexity on the comprehension of program modules*," Int. J. Man-Machine Stud., vol. 21, pp. 91–103, 1984.

[15] E. R. Iselin, "*Conditional statements, looping constructs, and program comprehension: an experimental study*," Int. J. Man. Mach. Stud., vol. 28, no. 1, pp. 45–66, Jan. 1988.

[16] B. Curtis, "*Substantiating Programmer Variability*," in Proceedings of the IEEE, 1981, vol. 69, no. 7, p. 846.

[17] M. Klerer, "*Experimental study of a two-dimensional language vs Fortran for first-course programmers*," Int. J. Man. Mach. Stud., vol. 20, pp. 445–467, 1984.

[18] S. Ajami, Y. Woodbridge, and D. G. Feitelson, "*Syntax, Predicates, Idioms - What Really Affects Code Complexity?*," in 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC), 2017, vol. 24, no. 1, pp. 66–76.

[19] D. I. De Silva, N. Kodagoda, S. R. Kodituwakku, and A. J. Pinidiyaarachchi, "*Analysis and enhancements of a cognitive based complexity measure*," in 2017 IEEE International Symposium on Information Theory (ISIT), 2017, pp. 241–245.

[20] U. Chhillar and S. Bhasin, "*A New Weighted Composite Complexity Measure for Object-Oriented Systems,"* Int. J. Inf. Commun. Technol. Res., vol. 1, no. 3, pp. 101–108, 2011.

[21] L. Jain and S. Singh, "*A journey from cognitive metrics to cognitive computers,"* IJARET, vol. 4, no. 4, pp. 60–66, 2013.

[22] V. R. Basili, "*The role of experimentation in software engineering: past, current, and future*," in Proceedings of IEEE 18th International Conference on Software Engineering, 1996, pp. 442–449.

[23] V. Rajlich and G. S. Cowan, "*Towards standard for experiments in program comprehension,"* in Proceedings Fifth International Workshop on Program Comprehension. IWPC'97, 1997, pp. 160–161.

© 2019, IJCSE All Rights Reserved

Authors Profile

Ms Leena Jain is Associate Professor & Head(MCA)in Global Institute of Management and Emerging Technologies, Amritsar. She has done her doctrate in Computer Science and engineering from Punjabi University Patiala. Her doctoral thesis on 'Industrial scope of 2D packaging'. She has gided one Doctorate thesis under her guidance and is guiding four more students . besides this she She has 15 years of teaching experience and 10 years of Research Experience.

Mr Satinderjit Singh is working as a Associate professor and Head , Department of computer Applications GGNIMT, Ludhiana. He is research scholar in PTU, Kapurthala, Punjab, India. He is persuing his Ph.d in computer Science and Engginering. He has a 18 years of teaching experience.

.

© 2019, IJCSE All Rights Reserved