International Journal of Computing and Applications | **ISSN: 0973-5704**
Volume 13, Number 2, (July-December 2018), pp 221-225
© Serials Publications, New Delhi (India)

OPTIMAL COMPONENT SELECTION FOR ELICIT MODEL USING COMPLEXITY METRICS

Kavita[1], Gaurav Aggarwal[2] and Neha Sharma[3]
[1-3]Manipal University Jaipur.
Email: [1]Kavita.jhajharia@jaipur.manipal.edu, [2]Gaurav.aggarwal@jaipur.manipal.edu, [3]nehav.sharma@jaipur.manipal.edu

Abstract: In Component Based Development (CBD) systems are built from existing components, primarily by assembling and replacing components. Software component has been developed vigorously. Thus they are reliable and the reason is that these components are tested under several of situations before use. The software development techniques changed over time from traditional to Component-Based Software Engineering (CBSE) many issues should be resolved and complexity of component is also an issue to be resolved. The main intend of this paper is to calculate the complexity of the component and opt the best component for system development through proposed algorithm.

Keywords: Component based software engineering, Software metrics, Complexity, Component.

1. INTRODUCTION

As today's software development growing rapidly in size, the prophecy of software reliability plays vital role in process of software development. CBSE give emphasis to reusability, that is, the construction and reuse of software building blocks. These blocks are often known as components.

CBSE approach is based on the principle of developing a software by opting appropriate components and assemble them by a well-defined software architecture. CBSE is used to develop/ assemble software from existing components. It is concerned with composing, selecting and designing components [11].

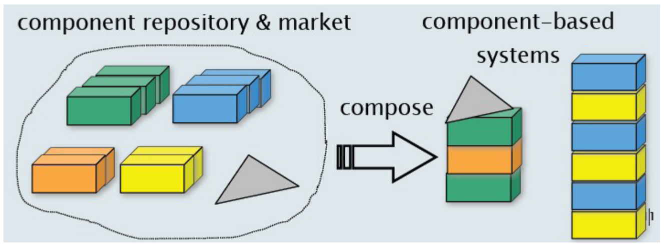


Figure 1: Component Based Software Engineering

The use of CBD requires good modular design. This modularity provides quality properties like:

- Comprehensibility/understandability
- Maintainability
- Flexibility

The Reusability is the pillar of CBSE and the reusability provides software developers with a number of assessable benefits [10].

Components: Component is an independently deliverable package of operations i.e. independently deployable and subject to composition by third parties [4].

There are three types of components:

1. *Off-the-shelf components:* The cost for attainment and assembling of off-the-shelf components will nearly always be less than the cost to develop corresponding software and risk is quite low.
2. *Full-experience components:* The risk associated with adaptation and integration is usually tolerable.
3. *Partial-experience components:* The cost to adapt partial-experience components can be more than the cost to develop a component from scratch and the risk will also be high [2].

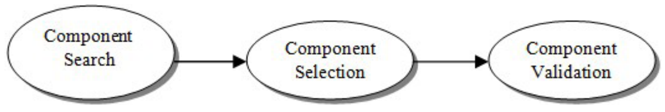


Figure 2: Component Selection Process

Incongruously, reusable software components are often unkempt during planning but becomes a supreme concern during the development phase of the software.

Component Interfaces: The functionality of a component is defined by its interfaces. The services provided by any component are made available through an interface.

The component interface is of two types:

1. Required Interface: specify the functionalities and services that a component requires to carry out its tasks.
2. Provided Interface: specify the functionalities and services that a component can offer.

CBSE have two activities of software process development:

Domain Engineering: Domain Engineering (DE) performs the work required to establish a set of software components to be reused by the software developer. DE is intended to ameliorate the quality of developed software through reuse of software components. This approach emphases on developing component-based software systems by selecting from off-the-shelf components and assembling that set of selected components within an appropriate software architecture [3].

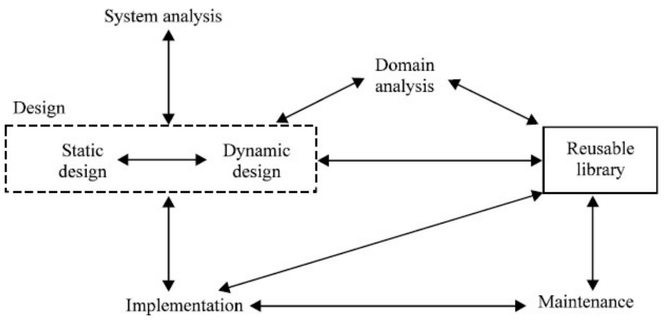


Figure 3: Domain Engineering

DE includes three major activities:

1. **Analysis:** Domain analysis mainly produces a domain model that represents the common and varying properties of application within the domain.
2. **Design:** Domain design takes the domain model produced throughout the domain analysis phase and aims to produce a basic architecture to which all application within the domain can conform.
3. **Implementation:** Domain Implementation is the formation of a process and tools for competently producing a customized program in the domain [5].

Component Based Software Engineering

CBSE approach built the software by using pre-existing components and assembling them into a software. This approach doesn't only reduces the cost and time but also accelerate the speed of development with better quality.

Cyclomatic Complexity

As described by the McCabe, a main purpose of this metric is to identify software modules that will be difficult to test or maintain. The complexity of a code can be measured by control the number of paths in program. McCabe introduced that the complexity is measured through the paths that can lead the program to execution [9].

Using graph theory as a mathematical explanation, pick a directed graph, G, has pair (V, E). Where V is vertex set of graph and its elements are called vertices. E is called the edge set of graph and its elements are called edges. Hence, the definition in graph theory of the cyclomatic number, V, comes in the form

$$V(G) = e - n + 2$$

where,

V(G) = the cyclomatic complexity,

e = the number of edges,

n = the number of nodes.

This formulation defines the cyclomatic complexity [1]. Several properties of cyclomatic complexity are stated below:

(I) V(G) always be greater than equal to 1.

(II) V(G) is the maximum number of linearly independent paths in G; it is the size of a basis set.

(III) Insertion and deletion of functional statements to G does not affect V(G).

(IV) G has only one path if and only if V(G) = 1.

(V) Insertion of a new edge in G increases V(G) by unity.

(VI) V(G) is dependent only on decision structure of G[6].

2. PROPOSED ALGORITHM

Step 1: Begin

Step 2: Select (SR, CR, C, R, CC) SR = System Requirements

CR = Component Requirements

C = Component

R = Repository

CC = Cyclomatic Complexity

Step 3: R = {1 to X}

//X is the number of components Repository (R)

Step 4: Select C from R If SR = CR

1. Check CC of C
2. Compare CC of every corresponding component
3. Select C with less CC

//In Repository (R) if more than one Component Requirement (CR) fulfils System Requirements (SR), then check the component complexity of every Component (C) and opt the component with less Cyclomatic Complexity (CC).

Step 5: Else if SR ≠ CR

//The System Requirement (SR) doesn't match with Component Requirement (CR), then

1. Alter SR.

   Repeat step 1 to 4.

Step 6: End.

The proposed algorithm selects the components through elicit model to get component with less complexity. The result shows the cyclomatic complexity calculation of component. The less complex component leads in high performance and a reduced amount of runtime.

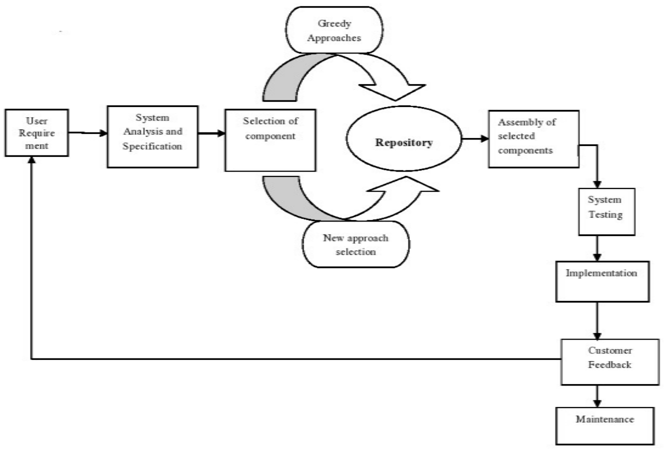


Figure 4: **The Elicit model**

3. RESULT

The following results are of the cyclomatic complexity calculation of component to select better component.

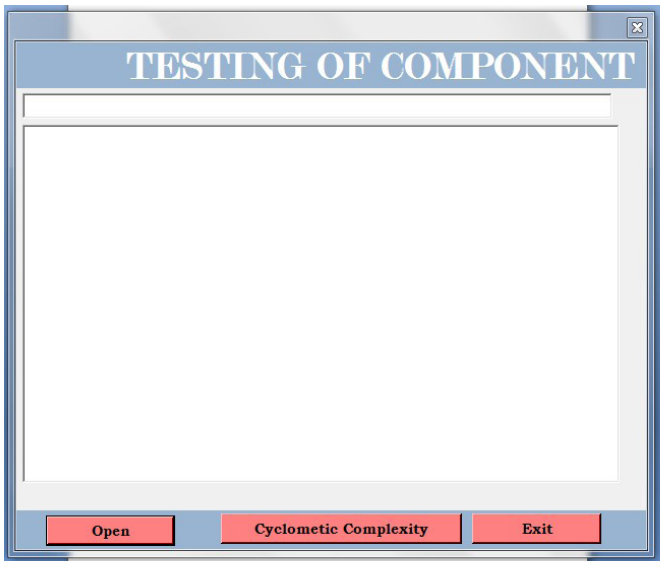
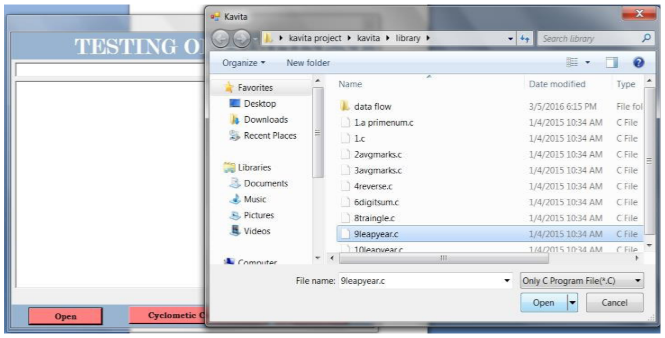


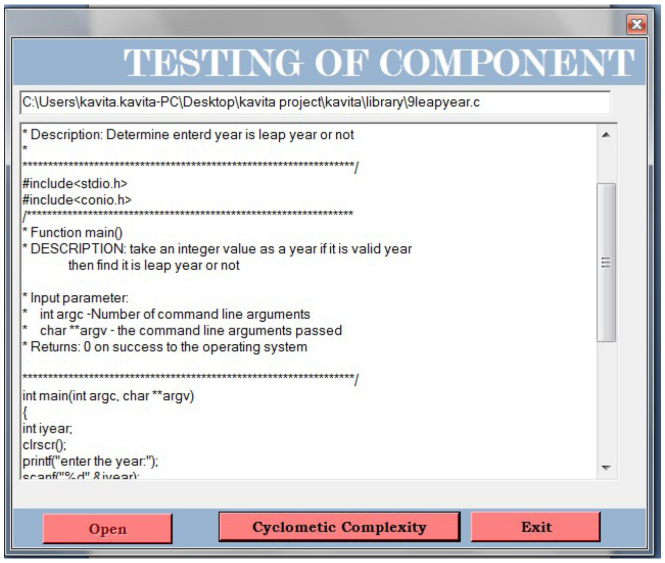
Figure 5: **Component Testing Software**



```
Name                Date modified        Type
data flow           3/5/2016 6:15 PM     File fol
1.a primenum.c      1/4/2015 10:34 AM    C File
1.c                 1/4/2015 10:34 AM    C File
2avgmarks.c         1/4/2015 10:34 AM    C File
3avgmarks.c         1/4/2015 10:34 AM    C File
4reverse.c          1/4/2015 10:34 AM    C File
6digitsum.c         1/4/2015 10:34 AM    C File
8traingle.c         1/4/2015 10:34 AM    C File
9leapyear.c         1/4/2015 10:34 AM    C File
10leapyear.c        1/4/2015 10:34 AM    C File

File name: 9leapyear.c          Only C Program File(*.C)
```

Figure 6: **Component Selection Process**



```
C:\Users\kavita.kavita-PC\Desktop\kavita project\kavita\library\9leapyear.c
```

```
* Description: Determine enterd year is leap year or not
*
*****************************************************************/
#include<stdio.h>
#include<conio.h>
/****************************************************************
* Function main()
* DESCRIPTION: take an integer value as a year if it is valid year
           then find it is leap year or not

* Input parameter:
*    int argc -Number of command line arguments
*    char **argv - the command line arguments passed
* Returns: 0 on success to the operating system

*****************************************************************/
int main(int argc, char **argv)
{
int iyear;
clrscr();
printf("enter the year:");
scanf("%d",&iyear);
```

Figure 7: Leap Year Calculation Program Inserted

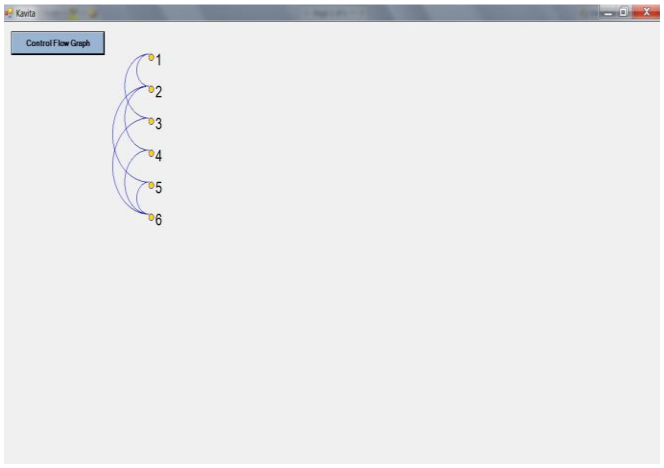
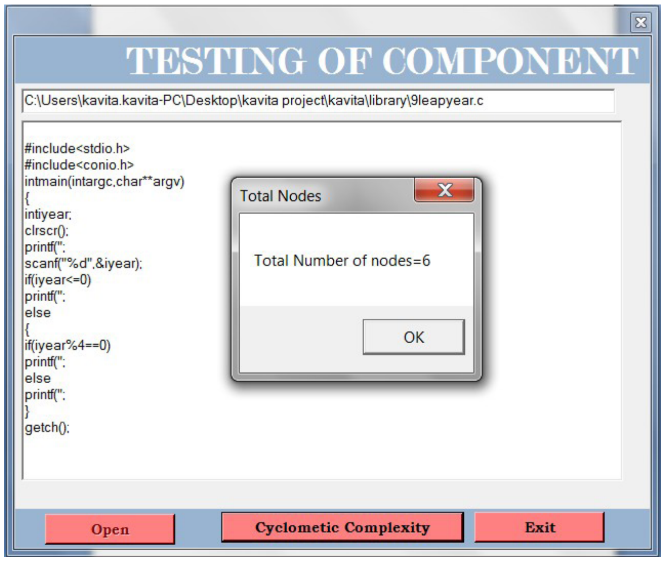


Figure 8: Control Flow Graph of Component

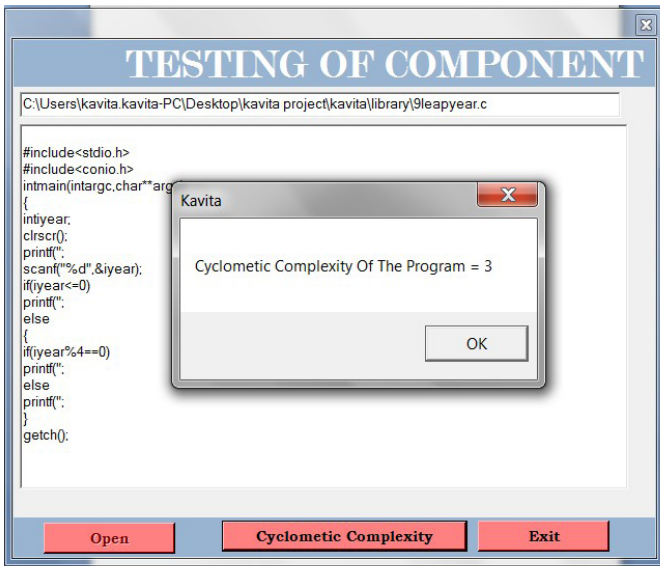


```
C:\Users\kavita.kavita-PC\Desktop\kavita project\kavita\library\9leapyear.c
```

```
#include<stdio.h>
#include<conio.h>
intmain(intargc,char**argv)
{
intiyear;
clrscr();
printf(";
scanf("%d",&iyear);
if(iyear<=0)
printf(";
else
{
if(iyear%4==0)
printf(";
else
printf(";
}
getch();
```

Total Number of nodes=6

Figure 9: Nodes of Component



```
C:\Users\kavita.kavita-PC\Desktop\kavita project\kavita\library\9leapyear.c
```

```
#include<stdio.h>
#include<conio.h>
intmain(intargc,char**arg
{
intiyear;
clrscr();
printf(";
scanf("%d",&iyear);
if(iyear<=0)
printf(";
else
{
if(iyear%4==0)
printf(";
else
printf(";
}
getch();
```

Cyclometic Complexity Of The Program = 3

Figure 10: Cyclomatic Complexity of Component

4. CONCLUSION

CBSE is known for improved productivity and quality; this can be achieved by using prefabricated components. Currently, the trend is towards an exponential increase of COTS on the market place. Cyclomatic complexity is a software metric used to measure the complexity of software component. Complexity of a source code can be calculated to check whether the component would be appropriate or not for the specific software. The selection of component is tough task in CBSE. Less complexity component would be better to use. So in this paper the selection of optimal component is done through proposed algorithm, after section of component the complexity of the component is checked and the less complexed component is opted for better results.

Future Work: The calculation of this complexity is only useful on white box testing doesn't work on black box testing as internal working in black box testing is not accessible. This can be resolved in future research.

References

[1] McCabe T.J., "A Complexity Measure", *IEEE transactions of Software Engineering,* Vol. SE2, No. 4, 1976.

[2] Pressman R.S., "Software Engineering, A practitioner's approach", *McGraw-Hill Higher Education*, 5th Ed., ISBN 0-07-365578-3, 2001.

[3] Pour G., "Component-Based Software Development Approach: Is It the Next Silver Bullet?", *IEEE,* pp. 491-492, 1998.

[4] Koziolek H., "Performance evaluation of component-based software systems: A survey", *Performance Evaluation, Elsevier*, pp. 634-658, 2010.

[5] Basha N.M.J., Moiz S. A., "Component Based Software Development: A state of Art", *IEEE-International Conference On Advances In Engineering, Science And Management* (ICAESM -2012), pp. 599-604, 2012.

[6] Smith K.R., "Linux, Openbsd, Talisker: A Comparative Complexity Analysis", 1994.

[7] Tomar P.,Gill N.S., "New Algorithm for component selection to Develop Component-Based Software with X Model", *Lecture Notes on Software Engineering,* Vol. 1, No. 3, pp. 298-302, 2013.

[8] Madi A., Zein O.K., Kadry S., "On the Improvement of Cyclomatic Complexity Metric", *International Journal of Software Engineering and Its Applications,* Vol. 7, No. 2, 2013.

[9] Gill, Geoffrey K., Kemerer, Chris F., "Cyclomatic Complexity Metrics Revisited: An Empirical study of Software Development and maintenance*", Cambridge, Mass. : Center for Information Systems Research, Sloan School of Management, Massachusetts Institute of Technology*, CSIR WP No. 218, 1990.

[10] Vescan, A., Pop, H. F., "Constraint Optimization-based Component Selection Problem", *Studia Universitatis Babes-Bolyai, series Informatica*, Vol. 3, 2008, pp. 3-14, 2008.

[11] Crnkovic I., Larsson S., Chaudron M., "Component Based development process and component life cycle", *journal of computing anf information technology,* pp. 321-327, 2005.