EMPIRICAL EVALUATION OF METRICS FOR COMPONENT BASED SOFTWARE SYSTEMS

Abhikriti Narwal

1Lecturer, S.D.I.T.M, Israna, Panipat
E-mail: abhikrititin@gmail.com

Abstract- Component-based Software Systems (CBSS) have now become more generalized approach for application as it mainly focus on assembling individual components, to develop the application. Today’s applications are large, complex and are not integrated. Although they come packaged with wide range of features but most features can neither be removed, upgraded independently or replaced nor can be used in other applications. Today Component based software Development (CBSD) is getting accepted in industry as a new effective development paradigm. CBSD is capable of reducing development costs and improving reliability of an entire software system using components. It emphasizes the design & construction of software system using reusable components. In order to maximize the reuse of code there is a need to manage the increasing complexity in the software systems. Various metrics have been developed by researchers for improving the quality of software components as traditional software metrics are neither suitable nor sufficient in measuring complexity of these components. The paper will take a closer look at the awareness of usage of Complexity metrics for Component-based Software Systems in Software Industry. Most of the findings present in this paper are based on Survey and Interviews with Software developers, Testers, Research Engineers of many major Software Companies. Various suggestions and recommendations made in this paper if implemented properly will go a long way in improving the quality of Software in Software Industry.

Keywords: Component, Component-based Software Systems

INTRODUCTION

There is a trend of using components in Software application development, due to its obvious advantages low cost, high quality and less time to develop applications along with several others. The core of component-based software systems (CBSS), as known as component based software engineering (CBSE) is a kind of flexibility required by open systems to reuse software components. The pressure for reducing software development life cycles and costs has lead to an increasing interest in CBSS that not only facilitates the process of software development but also changes the way to develop software applications. Nowadays CBSS is getting accepted in the industry as a new effective software development paradigm [1]. Most of the CBSS research has been now inclined toward methods and approaches in the development and in comparison between various software systems [2].

What is complexity?

IEEE defines software complexity as “the degree to which a system or component has a design or implementation that is difficult to understand and verify” [3].

There are lot of other definitions also which have been proposed by many researchers some of them are as follows:

- Complexity as a measure of the resources expended by a system, while interacting with a piece of software to perform a given task. If the interacting system is a program, then complexity is defined by the difficulty of performing tasks such as coding, debugging, testing or modifying the software [4].

- Software Complexity is the difficulty to maintain, change and understand software [5].

- The complexity of an object is a measure of the mental effort required to understand and create that object [6].

- Complexity is a major cause of unreliability in software [7].

In conventional software development, complexity can be defined as the difficulty to analyse source code, modify and maintain its modules. However in CBD, due to its black box nature in which source code is not available, component may only be customised, added or removed. Customization can be performed through its interface methods and properties.
COMPLEXITY METRICS

Software Metrics are intended to measure the software quality and performance characteristics quantitatively, encountered during the planning and execution of software development. These can serve as measures of software products for the purpose of comparison, cost estimation, fault prediction and forecasting. Metrics can also be used in guiding decisions throughout the life cycle, determining whether software qualities imitative are financially worthwhile [8].

The Component Complexity is a measure of the inter-relationships between the statements in a software component [9].

There are different ways to present the relationships between the statements in a module in form of a graph. One of the most common graphs is the flow-graph, which shows the control flow in a module. Figure 2.1 presents a program segment, written in a higher level language, and its corresponding flow-graph.

1 L := 0;
2 Repeat
3 Readln(New);
4 If L < New
  then
5 L := New
6 Until New < 0;
7 Writeln(L);

![Flow-graph of the program segment](image)

Fig 2.1: A program segment and its corresponding flow-graph

The white dots in the flow-graph represent the start and the stop node. Black dots represent different statements in the program segment. Some of the interesting component complexity metrics are:

(a) **Cyclomatic Complexity**;
(b) **Number of Logical Operators**;
(c) **Essential Cyclomatic Complexity**;
(d) **Myer’s Interval**;
(e) **Maximum Nesting of Control Structures**;
(f) **Estimated Static Path Count**

Other component complexity metrics that appear to be correlated with software quality, but are too complex to explain in brief are:

- **Prather’s Metric**;
- **The Lambda Metric**;
- **The YAM Metric**;
- **The Basili-Hutchens Metric**;
- **The Nao Metric**;
- **Testbed Structure Metric**;
- **The VINAP Metric**.

Component complexity metrics—express the inter-relationships between the statements in a software component. They can be grouped in a number of ways depending on the goal that you want to achieve with them.

CASE STUDY DESIGN

In a case study one situation is usually compared with another. It can be organized as a sister project, baseline, or random selection.

1. **Sister project**: Two projects selected, called sister projects are compared to each other. Each of them is typical for the organization and has similar values for the state variables that are going to be measured.

2. **Baselines**: A project is compared to a baseline. Data is gathered from various projects in the organization, regardless of how different they are from each other. Then a measure of the central tendency and dispersion of the collected data is calculated. That presents an average situation, a typical situation in the company, a baseline.

3. **Random selection**: A single project is partitioned into parts. Then for example one part uses some new technique and others do not.

FORMAL EXPERIMENT

In short, a formal experiment is a rigorous, controlled investigation of an activity, where key factors are identified and manipulated to see the effect on the outcome. So in this paper I have conducted the formal experiment by using Matlab software in which results have been computed and the comparison of results have been made between various components so that component with least complexity get selected so that testers and developers will require less time and less vulnerability to develop the software.

CASE STUDY

Suppose we have a CBSS that consists of four components as Fig. 4.11. We know that there are some interactions between component A and B (both directions),
from A to C (single direction), from A to D (single direction), between B and D (both directions) and from D to C (single direction). The relative data are assumed among four components as Table 4.1

Two data for MV<sub>j,i</sub> in Table 4.1 are the numbers of methods and instance variables in class C<sub>i</sub> invoked by class C<sub>j</sub>. According to Fig. 4.11 and Table 4.1, we may calculate the coupling and interface metrics on the levels of the individual component CBSS.

**INVESTIGATION OF METRICS OF VARIOUS COMPONENTS.**

In this section we will find out various values of metrics for the components by using the above Case study. So values of various metrics have been calculated as below.

- **COUPLING METRICS FOR THE COMPONENT AND SYSTEM**
  According to (1), we will get the coupling metric for each component in CBSS, shown in Table 4.2

- **COHESION METRIC FOR COMPONENT AND SYSTEM**
  According to (2), we will get the cohesion metric for each component in CBSS, shown in Table 4.3

- **INTERFACE METRICS FOR THE COMPONENT AND SYSTEM**
  According to the definitions for CIM and AIM, see (3), we will have the interface metric for each component in CBSS, shown in Table 4.4

From (7), system actual interface metric (SAIM) will be:

\[ SAIM = \frac{\sum_{j=1}^{m} AIM_j}{m} = 1.18 \]

**COMBINED SOLE SYSTEM COMPLEXITY METRIC**

If we give the weights for system coupling, cohesion and interface metrics as 0.5, 0.2 and 0.3, from (8), we will get the SSCM of the exampled CBSS as:

\[ SSCM = \alpha * SCOUP + \beta * SCOH + \gamma * SAIM = 0.92 \]
**COMBINED SOLE COMPONENT COMPLEXITY METRIC**

Let’s go back to compute the sole component complexity metric (SCCM) for CBSS. From Table 4.2 to Table 4.4, we will have SCCM metrics for each component as Table 4.5 if we assign the coupling, cohesion and interface weights as 0.5, 0.2 and 0.3, referring to (8).

<table>
<thead>
<tr>
<th>Component</th>
<th>ML_j</th>
<th>CM_j</th>
<th>AI_j</th>
<th>SCCM_j</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.31</td>
<td>0.48</td>
<td>1.20</td>
<td>1.129</td>
</tr>
<tr>
<td>B</td>
<td>1.22</td>
<td>0.25</td>
<td>1.13</td>
<td>0.999</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0.33</td>
<td>0.87</td>
<td>0.927</td>
</tr>
<tr>
<td>D</td>
<td>1.27</td>
<td>0.83</td>
<td>1.47</td>
<td>1.242</td>
</tr>
</tbody>
</table>

Table 4.5 Complexity metrics for the Component

4. CONCLUSION FROM THE CASE STUDY

So after calculating various values for the metrics the following conclusions have been made out from the case study:

1. We know that the component D has the highest complexity metric and needs the most investment such as time/cost, while component C has the lowest complexity value. The complexity metric of component B is less than that of component A.

2. Similarly as above comparisons of complexity metrics among the components, the system complexity data such as 0.95 (SCOUP), 0.47 (SCOH), 1.18 (SAIN) and 0.92 (SSCM) have hardly any meaning for themselves. However, when such data are used to compare the complexity levels among several software systems, the developers will know which CBSS needs more people and more time during the coding and testing stages, or they may expect the vulnerabilities will happen in which component according to the complexity metrics.

3. So C component is more reusable than the other elements as the complexity of this element is minimum among all the other components so the component will require the less effort by the developers and testers while implementing the software systems. It’ll get reuse again and again and less error will get propagate in software systems.

RESULTS AND CONCLUSIONS

Results calculated after doing implementation of the above metrics will be represented by the various graphs which are shown as
So after calculating various metrics value for each component and in the last by using all the values and by calculating the Combined Sole Component Complexity Metric the graph comes out to be:

![Combined Sole System Complexity Metric for Various Components](image)

**CONCLUSION**

We have calculated the Complexity Metric for software components and presented. This metric is based on the complexities involved in interface methods and properties of these interfaces in components. Results show that complex components take much time to execute than simple components and are very difficult to maintain. Also, complex components are not easily understandable. These facts may be obvious, though, but our work has brought quantitative affirmation of all these.

So Conclusions made from these results are:

1. The pressure for reducing software development life cycles and costs has lead to an increasing interest in CBSS. In order to manage the software increasing complexity and to maximize the reuse of code, adequate metrics to quantify component quality on the levels of both individual component and system are necessary.

2. The survey and case study shows that the system which has less complexity is easier to reuse. Reusability may have several direct or indirect factors like cost, efforts, and time.

Upon the research on the classical evaluation measures for software systems, we argue that the traditional metrics that mainly rely on the lines of codes are not suitable for CBSS. Therefore we provide an account of novel security measures for component and CSS by adequate coupling, cohesion and interface metrics in this thesis. Our focuses are to evaluate both individual component and assembly relation between components at the design stage of CBSS development life cycle. We believe that our efforts may help to manage the complexity of CBSS and to validate the component/system at a very early stage in the software development process. The case study shows that the methods presented in the thesis are effective by comparing the complexity metrics for each component. We believe the complexity measures for CBSS must be also useful in a similar way. People never keep into considerations the metrics. Rest uses metrics sometimes according to their need.

**FUTURE WORK**

The software metrics research for CBSS is still in the early stages in the long run. As CBSS is an emerging we will continue our effort in software metrics for CBSS such as

- To optimize the coupling metric by combining indirect coupling into direct coupling. More empirical research by applying our novel metrics in the real CBSS systems is also one of our future works.
- Reusability metrics can be keeping into considerations for component–based systems. These metrics can be used for further study and empirical validation can be performed and such new enhanced metrics can be proposed.

**References**


