Journal of Science and Technology
ISSN: 2456-5660 Volume 5, Issue 2, March-April, 2020, PP 71-75
www.jst.org.in

Measurement of Dynamic Complexity using Aspect Oriented Approach

Manju[1], Pradeep Kumar Bhatia[2]
[1]*(Research Scholar, Guru Jambheshwar University of Science and Technology, Hisar,India)*
[2]*(Professor, Guru Jambheshwar University of Science and Technology, Hisar,India)*

Abstract: *Complexity plays a vital role to find quality of software. There are lots of static or compile time metrics available in literature to find complexity of software, but very few metrics are available to find runtime complexity of software. In this paper, value of existing complexity metric is calculated by using CodeMR tool which is an Eclipse plugin. Further, we introduced a new metric named DWMC, dynamic weighted method complexity to measure complexity of a class at run time. A new tool is purposed named DynaDwmc implemented in AspectJusing aspect oriented programming to measure the value of purposed metric. After that, comparison of static WMC and runtime DWMC metric values is done. An experimental study is done on 15 java classes and it is concluded that purposed metric DWMC plays a significant role to find complexity, hence quality of software system.*

Keywords*: *Dynamic, complexity, quality, aspect

Received on: 05-01-2020 Revised on: 15-02-2020 Published on: 09-04-2020

I. Introduction

Metrics at any given time during the project execution indicate the quality of the product being built. Metric analysis for similar projects execution gives the capability levels of processes being used over a period of time. Metric are simple to defined, easy to understand,robust to use and able to improve.Chidamber and Kemerer (CK) have introduced six metrics in his suite[3]. Weighted method per class(WMC) is one of the metric in his suite. It is defined as the weighted sum complexity of all the method defined with in a class. If all the methods in class have complexity as unity then WMC will be equal to number of methods. i.e.n .

$$WMC = \sum_{i=1}^{n} c_i$$

Where c_1,c_2….$c_n$be the complexity of the methods $m_{1....}m_n$ defined within the class.In this paper,we extract the value of WMC metric by using CodeMR[9] tool.CodeMR is mainly a Eclipse plugin[10] to find the quality of software in terms of 3 main quality measures i.e. complexity, cohesion and coupling for java,C++ and scala languages.It also provides Graph and Dependency views of the metrics calculated by the tool. For dynamic code analysis[4] we purposed a new tool implemented in AspectJ[1], implemented in java using aspect oriented programming.There are some advantages of aspect oriented approach[8] over the other approaches to trace the events at runtime.

1. The process of building a tracing or profiling frame work using aspect oriented approach is relatively simpler than any other approach.
2. This approach does not generate a large amount of data.
3. This approach produces results which are valid among all programme execution environment.
4. This approach is relatively in expensive and much more practical in nature.

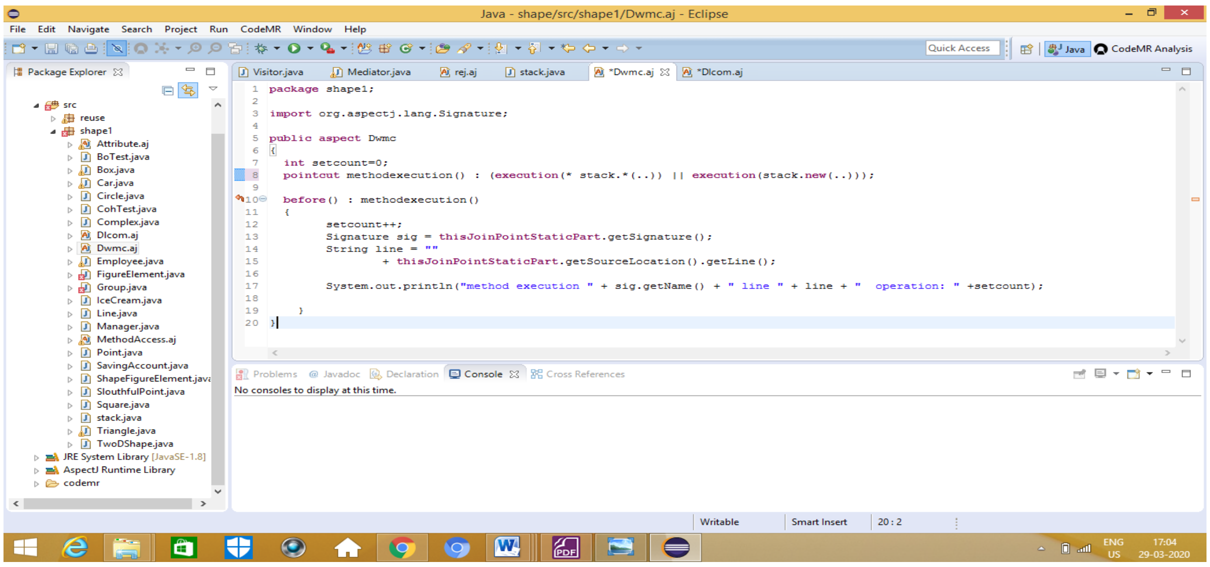
II. ProposedMetric and Tool

DWMC (Dynamic Weighted method complexity): It is defined as the number of times a method is executed at runtime with in a class. If all the method of a class is executed at runtime then static method complexity and runtime weighted method complexity are same for that class.

$$DWMC = \sum_{i=1}^{n} \mu_i$$

Where μ_i is the number of times method iis executed at runtime and n is the total number of methods within a class.

DynaDwmc is a tool to calculate the purposed metric.Tool is implemented in AspectJ. AspectJ is mainly an implementation of Aspect Oriented programming in java.Figure 1. shows the coding of aspect in AspectJ which is to be run with any class to find the value of DWMC metric.



```
package shape1;

import org.aspectj.lang.Signature;

public aspect Dwmc
{
  int setcount=0;
  pointcut methodexecution() : (execution(* stack.*(..)) || execution(stack.new(..)));

  before() : methodexecution()
  {
        setcount++;
        Signature sig = thisJoinPointStaticPart.getSignature();
        String line = ""
              + thisJoinPointStaticPart.getSourceLocation().getLine();

        System.out.println("method execution " + sig.getName() + " line " + line + "  operation: " +setcount);

    }
}
```

```
No consoles to display at this time.
```

Figure 1.Working ofDynaDwmc tool in AspectJ

III. Case Study

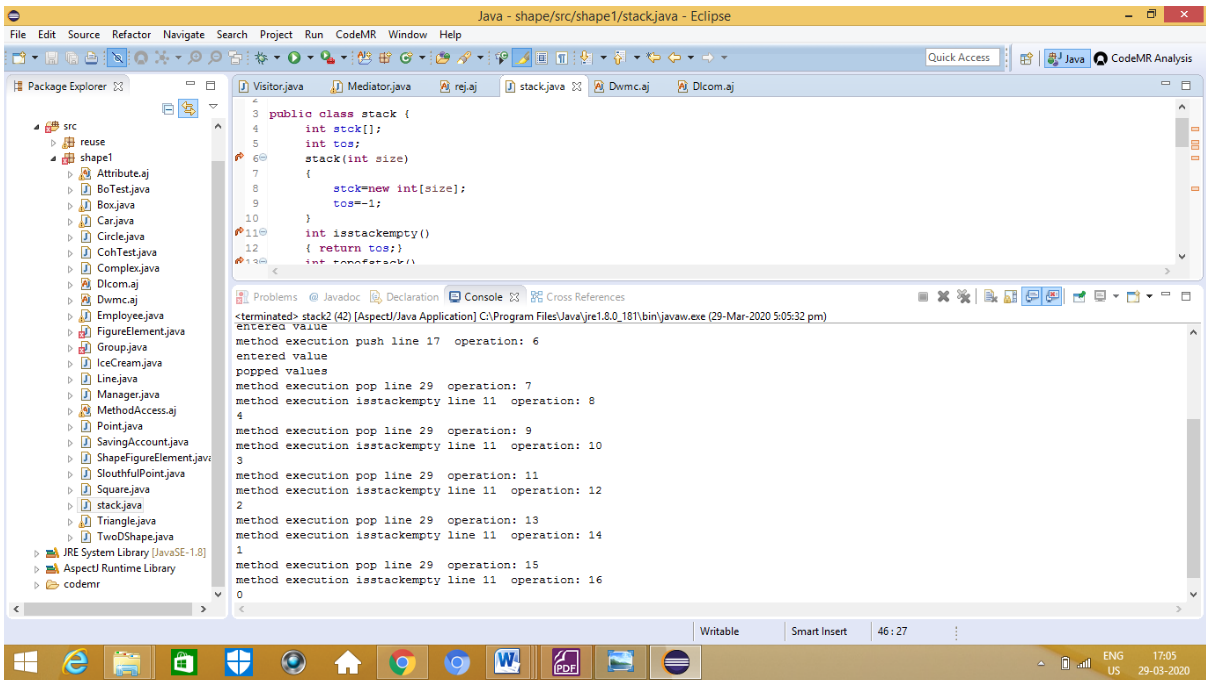
Consider a program written in java shown in Figure. 2. In this program,stack class contains 2 attributes and 5 methods named push,pop,isstackempty, topofstack and stack class constructor.

```
publicclass stack {
intstck[];
inttos;
stack(intsize)
    {
            stck=newint[size];
            tos=-1;
    }
intisstackempty()
{ returntos;}
inttopofstack()
    {
            returnstck[tos-1];
    }
void push(intitem)
{
if(tos==(stck.length-1))
            System.out.println("stack is full");
else
   {
            System.out.println("entered value");

tos=tos+1;
            stck[tos]=item;
   }
}
int pop()
 {
```

```
          if(isstackempty()==-1)
           {
                    System.out.println("stack overflow");
                    return 0;
           }
          else
                    returnstck[tos--];
 }
}
class stack2
{
publicstaticvoid main(String str[])
{
          stacks1=new stack(5);
          System.out.println("enter values in stack");
          for(inti=0;i<5;i++)
                    s1.push(i);
          System.out.println("popped values");
          for(inti=0;i<5;i++)
                    System.out.println(s1.pop());
}
}
```

Value of the WMC metric for this class is 5 as the number of methods in the class is 5 but CodeMR tool calculate its value 7 that means complexity of two methods is not considered as unity.If complexity of all the methods is considered as unity than it would be 5. To calculate the value of DWMC,DynaDwmc tool is used. Dwmc.aj aspect is created in AspectJ and run with stack.java class without interrupting the functioning of Stack class and the value of DWMC metric is calculated from runtime log as shown in Figure 3. Calculated value of DWMC metric is 16. So there is big difference between static and runtime weighted method complexity.



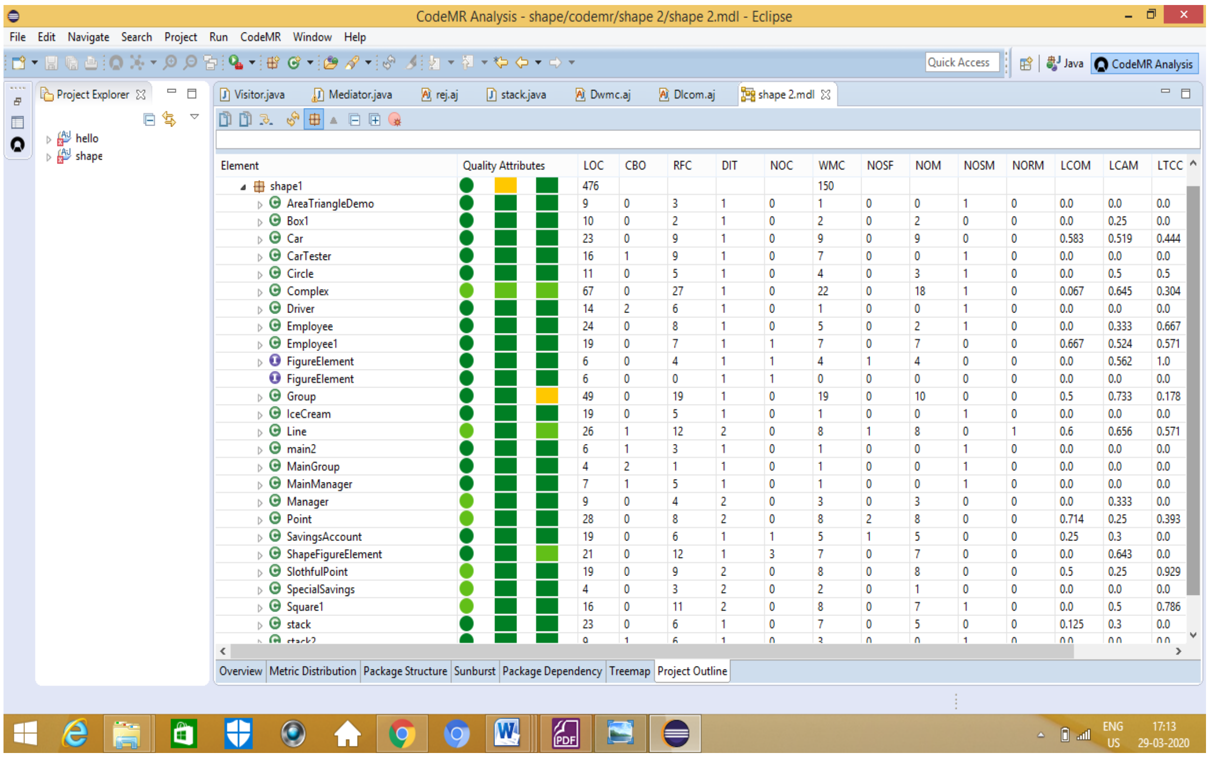
```
 3 public class stack {
 4      int stck[];
 5      int tos;
 6      stack(int size)
 7      {
 8          stck=new int[size];
 9          tos=-1;
10      }
11      int isstackempty()
12      { return tos;}
```

```
<terminated> stack2 (42) [AspectJ/Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (29-Mar-2020 5:05:32 pm)
entered value
method execution push line 17  operation: 6
entered value
popped values
method execution pop line 29  operation: 7
method execution isstackempty line 11  operation: 8
4
method execution pop line 29  operation: 9
method execution isstackempty line 11  operation: 10
3
method execution pop line 29  operation: 11
method execution isstackempty line 11  operation: 12
2
method execution pop line 29  operation: 13
method execution isstackempty line 11  operation: 14
1
method execution pop line 29  operation: 15
method execution isstackempty line 11  operation: 16
0
```

Figure 2.Event Tracing using DynaDwmc tool

IV. Experimental Study

An experimental study is conducted for the purposed metrics using 15 java classes.Classes are taken from web[12].CodeMR tool is used to calculate value of WMC metric for these classes as shown in Figure.4 and DynaDwmc tool is used to calculate values for purposed metric.



Element	Quality Attributes	LOC	CBO	RFC	DIT	NOC	WMC	NOSF	NOM	NOSM	NORM	LCOM	LCAM	LTCC
shape1	🟢 🟨 🟩	476					150							
AreaTriangleDemo	🟢 🟩 🟩	9	0	3	1	0	1	0	0	1	0	0.0	0.0	0.0
Box1	🟢 🟩 🟩	10	0	2	1	0	2	0	2	0	0	0.0	0.25	0.0
Car	🟢 🟩 🟩	23	0	9	1	0	9	0	9	0	0	0.583	0.519	0.444
CarTester	🟢 🟩 🟩	16	1	9	1	0	7	0	0	1	0	0.0	0.0	0.0
Circle	🟢 🟩 🟩	11	0	5	1	0	4	0	3	1	0	0.0	0.5	0.5
Complex	🟢 🟩 🟩	67	0	27	1	0	22	0	18	1	0	0.067	0.645	0.304
Driver	🟢 🟩 🟩	14	2	6	1	0	1	0	0	1	0	0.0	0.0	0.0
Employee	🟢 🟩 🟩	24	0	8	1	0	5	0	2	1	0	0.0	0.333	0.667
Employee1	🟢 🟩 🟩	19	0	7	1	1	7	0	7	0	0	0.667	0.524	0.571
FigureElement	🟢 🟩 🟩	6	0	4	1	1	4	1	4	0	0	0.0	0.562	1.0
FigureElement	🟢 🟩 🟩	6	0	0	1	1	0	0	0	0	0	0.0	0.0	0.0
Group	🟢 🟩 🟨	49	0	19	1	0	19	0	10	0	0	0.5	0.733	0.178
IceCream	🟢 🟩 🟩	19	0	5	1	0	1	0	0	1	0	0.0	0.0	0.0
Line	🟢 🟩 🟩	26	1	12	2	0	8	1	8	0	1	0.6	0.656	0.571
main2	🟢 🟩 🟩	6	1	3	1	0	1	0	0	1	0	0.0	0.0	0.0
MainGroup	🟢 🟩 🟩	4	2	1	1	0	1	0	0	1	0	0.0	0.0	0.0
MainManager	🟢 🟩 🟩	7	1	5	1	0	1	0	0	1	0	0.0	0.0	0.0
Manager	🟢 🟩 🟩	9	0	4	2	0	3	0	3	0	0	0.0	0.333	0.0
Point	🟢 🟩 🟩	28	0	8	2	0	8	2	8	0	0	0.714	0.25	0.393
SavingsAccount	🟢 🟩 🟩	19	0	6	1	1	5	1	5	0	0	0.25	0.3	0.0
ShapeFigureElement	🟢 🟩 🟩	21	0	12	1	3	7	0	7	0	0	0.0	0.643	0.0
SlothfulPoint	🟢 🟩 🟩	19	0	9	2	0	8	0	8	0	0	0.5	0.25	0.929
SpecialSavings	🟢 🟩 🟩	4	0	3	2	0	2	0	1	0	0	0.0	0.0	0.0
Square1	🟢 🟩 🟩	16	0	11	2	0	8	0	7	1	0	0.0	0.5	0.786
stack	🟢 🟩 🟩	23	0	6	1	0	7	0	5	0	0	0.125	0.3	0.0

Figure. 3. WMC metric values of java classes using CodeMR tool

The values of WMC and purposed metric of 15 java classes is shown in Table 1. From Table 1.we can say that there is huge difference between compile time and runtime weighted method complexity.Behaviour of every java class is different at compile time and run time.. Some classes like 1,8 and 13 shows same behaviour at both compile time and run time that means all the methods of these classes execute at runtime once. Classes like 2,3,6,7 and 11 shows huge difference between compile and run time values that depict method of these classes execute more than once at runtime and complexity of these classes increases at runtime. Classes like 10,14 and 15 have less value at runtime than compile time that depicts methods of these classes not execute at runtime hence complexity decreases at runtime and we can say class is not fully used at runtime.

Table 1.WMC and Purposed Metric DWMC values of 15 java classes

Sr. No.	Class Name	WMC	DWMC
1	Box	2	2
2	Stack	7	16
3	Car	9	24
4	Circle	4	3
5	Book	1	2
6	Complex	19	48
7	Employee	5	16
8	Ice Cream	1	1
9	Employee	7	9
10	Manager	3	2
11	Saving Account	5	10
12	Special Saving	1	2
13	Area Triangle	1	1
14	2D Shape	6	2
15	Square	8	4

V. Analysis Results

A new metric DWMC,Dynamic Weighted Method Complexity is purposed to calculate the complexity of a class at runtime. An experimental Study is done on 15 java classes and it is found that there is huge difference between values of compile time class complexity and run time class complexity as shown in Table 1.Hence, to measure the quality of software purposed metric plays a significant role by measuring the runtime complexity of a software.Analysis result of these classes is shown in Figure. 5. It Contains 15 java classes and compared values of WMC and purposed DWMC metric.

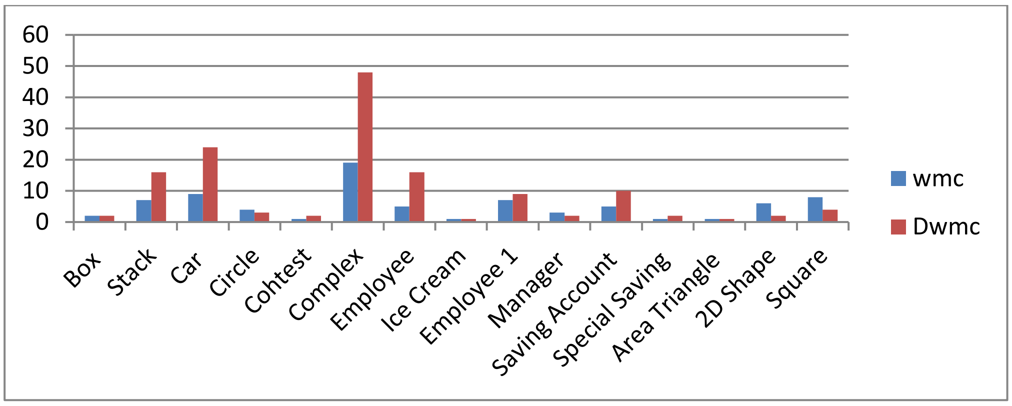


Figure 4.Comparison of WMC and DWMC metric

From Figure 5.We can easily say that complexity of a class increase at runtime in most of the cases. If we only consider static or compile time complexity of a class then we can't predict the actual behaviour of a class that affect the quality of software later and increases the maintainability cost. So purposed metric gives the actual behaviour of class and helps to decreases the cost of maintainability and improves quality of software. Statistical results are shown in Table 2.that shows correlationcoefficient between WMC and purposed metric is 0.90 at a significant level of 0.05.Statistical values are calculated by MATLAB tool.

Table 2. Statistical results on WMC and purposed metric

Sr.No.	Statistics Applied	DMC	DWMC
1.	Mean	5.26	9.46
2.	Median	5	3
3.	Min	1	1
4.	Max	19	48
5.	Standard Deviation	4.68	12.74

VI. Conclusion

CodeMR tool is used to calculate values of WMC (Weighted Method Complexity) metric from CK(Chidamber and Kemerer) metric suite at compile time. The purposed tool DynaDwmc is used to calculate values of purposed metric DWMC,Dynamic Weighted Method Complexity at runtime.An experimental Study is done on 15 java classes and it is found that there is big difference between values of compile time class complexity and run time class complexity. Therefore, purposed metric has its own existence. Hence, to measure the quality of software purposed metric plays a significant role by measuring the runtime complexity of software.

References

[1] AspectJ<http://www.eclipse.org/aspectj>
[2] S.R. Chidamber and C.F. Kemerer, "Towards a Metrics Suite for Object Oriented Design", *Proc. 6th ACM Conf. Object-Oriented Programming: Systems, Languages and Applications (OOPSLA)*, Phoenix, AZ, pp. 197-211, Oct.1991.
[3] S.R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object-OrientedDesign", *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, 1994.
[4]S. Yacoub, H. Ammar, and T. Robinson, "Dynamic Metrics for Object-Oriented designs", *Proc. 5th International Software Metrics Symposium*, Boca Raton, Florida, USA, pp.50–61,1999.
[5] J. Tian, and M.V. Zelkowitz, "A Formal Program Complexity Model and its Application", *Journal of Systems and Software*, vol. 17, no. 3, pp. 253-266,1992.
[6] D.P. Tegarden, S.D. Sheetz, D.E. Monarchi, "A Software Complexity Model of Object- Oriented Systems", *Decision Support Systems: the International Journal*, vol. 13, pp. 241-262,1995.
[7] J.C. Munson and T.M. Khoshgoftaar, "Measuring Dynamic ProgramComplexity",*IEEE Software*, vol. 9, no. 6, pp. 48-55, 1992.
[8] https://o7planning.org/en/10257/java-aspect-oriented-programming-tutorial-with-aspectj
[9] https://www.codemr.co.uk/docs/codemr-intellij-userguide.pdf
[10] https://www.eclipse.org/aspectj/doc/next/progguide/printable.html
[11]https://www.geeksforgeeks.org/