How to Realization Architectural testing model using Measurement Metrics

Lalji Prasad¹, Sarita Singh Bhadauria²

¹TRUBA College of Engineering & Technology/ Computer Science &Engineering, INDORE, INDIA
²MITS /Department of Electronics, GWALIOR, INDIA
Email: {¹lalji.prasad@trubainstitue.ac.in,²saritamits61@yahoo.co.in}

ABSTRACT

In this research paper we try to realize our architecture testing model [1], with the help of object oriented relationship, using different software metrics. Measurement is key element of any engineering process. Realization architectural testing tool, accomplish through measures, are used to better understand the attributes and relationship between the attributes of architecture that we create. In this paper, we use measurements to assess the quality of the architecture for testing of software product or the process used to build it. Instead, we attempt to derive a set of measurement that lead to metrics that provide an indication of the quality of some representation of software architecture, number of metrics has been defined for software architecture. These metrics try to capture different aspects of software architecture testing model and its process.


1. INTRODUCTION

The objective of ‘Design any Architectural Testing Tool’ is to facilitate a design that may contribute to the comprehensiveness of the software testing tool. Comprehensive means that it includes all or nearly all features and relationships required for migrating from one testing class to another. It is designed to overcome the limitation of existing software tools by providing a final class level architecture having relationships between various testing classes. Software quality is another focus of our architecture. We wish to achieve good maintainability, reusability, flexibility and portability in the architecture of the software testing tool by validating the architecture using testing algorithms and performing metrics calculation on each relationship existing between the different testing techniques [1][2][3].

2. ARCHITECTURAL TESTING TOOL REALIZATION (RELATIONSHIP) IN OBJECT ORIENTED DEVELOPMENT

Architectural diagram show figure1 in our previous paper [1], here we realize our architectural diagram or determine the quality of this architectural tool, through measuring its attribute and their relationship, by the help of software metrics. Here we categories our Architectural diagram into three parts first category Fault based Testing and Scenario based testing, second category Integration Testing and its derived classes and third category Functional Testing, Class based Testing and its derived classes[1][2][3]:

1. Fault based Testing and Scenario based Testing:

   The first category consists of fault-based testing and scenario-based testing [2].
   - The objective of fault-based testing within an OO system is to design tests that have a high likelihood of uncovering plausible faults. Because the product or system must conform to customer requirements, the preliminary planning required to perform fault-based testing begins with the analysis model. The tester looks for possible faults (i.e., aspects of the implementation of the system that may result in defects) [7].
   - When errors associated with incorrect specifications occur, the product does not do what the customer wants. Scenario-based testing concentrates on what the user does, not what the product does. This means capturing the tasks (via use-cases) that the user has to perform, then applying them and their variants as tests. Scenarios uncover interaction errors. Scenario-based testing tends to exercise multiple subsystems in a single test [8].

   Relationships:
   a) Uni-aggregation relationship between scenario based testing and fault based testing: Fault based testing is a part of scenario based testing and therefore a uni-aggregation relationship exists between these two testing’s.
   b) Composition relationship between scenario based testing and use based testing: Since, to perform scenario based testing use cases are designed and tested. Thus, use case based testing acts as a part of scenario based testing. Therefore, composition relationship exists between scenario based and use based testing.

2. Integration Testing and its derived classes:

   In the second category, integration testing is further divided into three parts: Thread based testing; cluster based testing and Use-based testing [3][6].
   - Thread-based testing integrates the set of classes required to respond to one input or event for the system. Each thread is integrated and tested individually [58].
• Use-based testing begins the construction of the system by testing those classes (called independent classes) that use very few (if any) server classes. After the independent classes are tested, the next layers of classes, called dependent classes, that use the independent classes are tested. This sequence of testing layers of dependent classes continues until the entire system is constructed [9].

• Cluster testing is one step in the integration testing of OO software. Here, a cluster of collaborating classes is determined by examining the CRC and object-relationship model [6].

Relationships:

a) Generalization/Inheritance relationship between integration testing and threads based testing, cluster based testing and use based testing:

Since, Integration testing is further divided into three parts-threads, cluster and use-based testing and therefore possess generalization relationship as integration testing class is generalized class for thread based, cluster based and use based testing.

b) Aggregation relationship between thread based testing and cluster based testing:

A cluster of collaborating classes have several threads in it and thus threads are part of clusters. But, existence of threads may exist independent of the clusters. Hence, aggregation which is one of the forms of association relationship exists between cluster and thread based testing.

c) Bidirectional navigability between Use based testing and thread based testing:

Use based testing begins the construction of the system by testing dependent and independent classes having threads. So testing can be done by designing use case for each thread firstly or after testing the threads, use based testing can be formed. Thus, bidirectional navigability exists between these two testing classes.

3. Functional Testing, Class based Testing and its derived classes:

The third category consists of functional testing, class based testing and its derived classes. This category is directly based on the requirements and specifications of software products. Partitioning-based testing and random testing are derived from class based testing and uses some properties of class based testing. Partition based testing is further classified into three types: State based testing, Attribute based testing and Category based testing [10].

Relationships:

a) Interdependency between functional testing and class based testing:

Inter dependence relationship exists between functional testing and class based testing as when functional specification are input for function level testing of any testing tools. Accordingly, functional specifications construct class based testing.

b) Generalization/Inheritance relationship between class based testing and partition & random based testing:

Class based testing is divided into two parts: partition based class testing and random based testing. Thus, their exists the generalization relationship as partition based class testing and random based testing are derived from class based testing and uses some properties of class based testing.

c) Generalization/Inheritance relationship between partition based testing and state based, attribute based and category based testing:

As partition based testing is further classified into state based, attribute based and category based testing and therefore, partition class has a generalization relationship with the other three testing classes respectively.

d) Aggregation relationship between state based testing and attribute based testing:

State based testing is associated with attribute based testing with its special form called as aggregation as each attribute possess some state and to perform attribute based testing, state based testing should be performed.

e) Composition relationship between category based testing and state based testing:

Category based testing possess composition relationship with the state based testing as for a particular category of a software testing, various objects may have various states and thus required to be tested based on their states.

f) Composition relationship between partition based testing and state based, attribute based and category based testing:

Since, state based, attribute based and category based testing can only be performed after partitioning the class. Thus, composition relationship exists between them as state based, attribute based and category based testing cannot exists without partition based testing.

g) Multiplicity relationship between partition based testing and state based, attribute based and category based testing:

Multiplicity between partition based testing and state based testing denotes that atleast 0 or 1 object of partition based testing is related with 1 or more objects of state based testing.
3. SOFTWARE METRICS USE IN REALIZATION OF ARCHITECTURAL TESTING TOOL

According above relationship among different testing technique/strategies, we realize architecture of testing tool using some software metrics and finally determine software quality of software. Chidamber, Agrawal and his colleague, [5,16] proposed twenty two metrics but, here used those metrics which are useful for my research work:

1. Size Metrics:
   a) Number of Attributes per Class (NOA) - It counts the total number of attributes defined in a class. NOA=$\sum_{i=1}^{n} a_i$ (where $i$=1 to $n$ and ‘a’ denotes attributes)
The nominal range for number of attributes in a class of project should range from 2 to 5. [13] NOA (> 10) probably indicates poor design. In our system 3/5 classes comes under the nominal range, with overall average of 5.4 not require further decomposition.

b) Number of Methods per Class (NOM)- It counts number of methods defined in a class[14].
NOM=∑ m_i (where i=1 to n and ‘m’ denotes methods)

c) Response For a Class (RFC) - The response set of a class (RFC) is defined as set of methods that can be potentially executed in response to a message received by an object of that class.
RFC=|RS_i|, is given by- 
RS = Response set of the class
M_i = set of all methods in a class (total n) and R_{ij} = {R_{ij}} = set of methods called by M_i.
The RFC is the "Number of Distinct Methods and Constructors invoked by a Class." The RFC for a class should usually not exceed 50 although it is acceptable to have a RFC up to 100. Refactor IT recommends a default threshold from 0 to 50 for a class [4]. In our project RFC satisfy the given range.

Weighted Methods per Class (WMC) - The WMC is a count of sum of complexities of all methods in a class.
To calculate the complexity of a class, the specific complexity metric that is chosen should be normalized so that nominal complexity for a method takes on value 1.0.
Consider a class C1, with methods M1, M2…, Mn. Let (Ij) = set of all instance variables used by method Mi. There are n such sets {I1},…….{In}. Let P = {(I1, Ij) | Ii ∩ Ij = 0} and Q = {(I1, Ij) | Ii ∩ Ij ≠ 0}.
If all n sets {(I1) … (In)} are 0 then P=0
LCOM = |P| - |Q|, if |P| > |Q| = 0 otherwise

3. Inheritance Metrics:

a) Depth of Inheritance - The depth of a class within the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree and is measured by the number of ancestor classes. In cases involving multiple inheritances, the DIT will be the maximum length from the node to the root of the tree.
DIT=∑ a_i (where i=1 to n and ‘a’ denotes no of levels from the root node)
DIT [12]. A value greater than 4 would compromise encapsulation and increase complexity.

b) Number of Children (NOC) - The NOC is the number of immediate subclasses of a class in a hierarchy. The upper and lower limits NOC of 1 and 3 correspond to a desirable average. This will not stop certain particular classes being the kind of utility classes which provide services to significantly more classes than three[11].

All of above metrics used for deciding completeness of software and provide help to measuring quality of software products.

4. ANALYSIS OF ARCHITECTURE OF TESTING TOOL USING METRICS CALCULATION

1. Composition relationship between fault and scenario based testing:
Figure 2: Relationship between fault and scenario based testing

a) Weighted methods per class (WMC)-
   If all method complexities are considered to be unity, then WMC = n, the number of methods in the class. Since, 4 member functions are used in this testing so WMC is 4.
   Similarly, in scenario based testing as 3 member functions, WMC is 3.

b) Number of Attributes per Class (NOA)-
   Number of Attributes (NOA) for fault and scenario based testing class is 1.
   So NOA = 1 for both the classes.
   Here NOA values follow our standard (2-5)

c) Number of Methods per Class (NOM)-
   Fault based testing has 4 methods so it’s NOM=4.
   Similarly, scenario based testing has 3 methods so it’s NOM=3.
   Here NOM values follow our standard (3-7)

d) Response for a class (RFC) –
   In scenario based testing runscenario method can invoke member functions of fault based testing that are wrongoperation(), unexpected(), incorrectinvocation().
   RS={scenario::runscenario}
   {fault based::wrongoperation, fault based ::unexpected, fault based ::incorrect invocation} Thus, RFC=4

Table 1: Composition Relationship between Fault Based Testing and Scenario Based Testing

<table>
<thead>
<tr>
<th>Testing relation and metrics values</th>
<th>WMC</th>
<th>NOA</th>
<th>NOM</th>
<th>RFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition Relationship between Fault Based Testing and Scenario Based Testing</td>
<td>4(FBT)</td>
<td>1(FBT)</td>
<td>4(FBT)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3(SBT)</td>
<td>1(SBT)</td>
<td>3(SBT)</td>
<td></td>
</tr>
</tbody>
</table>

2. Composition relationship between scenario and use case based testing:

Figure 3: Relationship between scenario and use case based testing

a) Number of Attributes per Class (NOA)-
   For scenario based, use based testing NOA=1.
   Here NOA values follow our standard (2-5)

b) Number of Methods per Class (NOM)-
   For scenario based NOM=4
   For use based NOM=3
   Here NOM values follow our standard (3-7)

c) The WMC for use case based testing class is 3 as it has 3 member functions.

d) Coupling between object classes (CBO) – It is the count of no. of other classes to which it is coupled.
   Here, scenario based testing is coupled to use case based testing so CBO for scenario based testing class will be 1.

e) LCOM- Lack of cohesion in the methods
   I1- {setup, cleanup, run scenario}
   I2- {setup}
   I3- {cleanup}
   I1∩I2 and I1∩I3 are not null but I2∩I3 is null.
   Let P {(I I ) | I I 0} and Q {((I I ) | I I 0} = , j i ∩ j = = , j i ∩ j ≠ i i .
   LCOM is 0 if numbers of null intersections are not greater than number of non-null Intersections.
   Hence LCOM in this case is 0 [P>Q]. Thus a positive high value of LCOM implies that classes are less cohesive. So a low value of LCOM is desirable.

Table 2: Composition Relationship between Use Case Based Testing and Scenario Based Testing

<table>
<thead>
<tr>
<th>Testing relation and metrics values</th>
<th>WMC</th>
<th>NOA</th>
<th>NOM</th>
<th>COB</th>
<th>LCOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition Relationship between Use Case Based Testing and Scenario Based Testing</td>
<td>3(UBT)</td>
<td>1(UBT)</td>
<td>3(UBT)</td>
<td>1(UBT)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3(SBT)</td>
<td>1(SBT)</td>
<td>3(SBT)</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

3. Inheritance between thread based, cluster based, use based and integration testing:
4. Composition between state based and category based testing:

![Diagram](image)

**Table 4:** Composition between State Based Testing and Category Based Testing

<table>
<thead>
<tr>
<th>Testing relation and metrics values</th>
<th>WMC</th>
<th>NOA</th>
<th>NOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition between State Based Testing and Category Based Testing</td>
<td>3(SBT)</td>
<td>1(SBT)</td>
<td>3(SBT)</td>
</tr>
</tbody>
</table>

5. Aggregation between thread and cluster based testing:

![Diagram](image)

**Table 3:** Inheritance between Use Case Based Testing, Thread Based Testing, Cluster Based Testing and Integration Based Testing.

<table>
<thead>
<tr>
<th>Testing relation and metrics values</th>
<th>WMC</th>
<th>NOA</th>
<th>NOM</th>
<th>DIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inheritance between Use Case Based Testing, Thread Based Testing, Cluster Based Testing and Integration Based Testing</td>
<td>3(UBT)</td>
<td>1(UBT)</td>
<td>3(UBT)</td>
<td></td>
</tr>
<tr>
<td>4(TBT)</td>
<td>1(TBT)</td>
<td>4(TBT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3(CBT)</td>
<td>1(CBT)</td>
<td>3(CBT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4(IBT)</td>
<td>1(IBT)</td>
<td>4(IBT)</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
6. Aggregation between state based and attribute based testing:

- **b) Number of Methods per Class (NOM)-**
  - For thread based NOM=4
  - For cluster based NOM=3
  - Here NOM values follow our standard (3-7)

- **c) Weighted method per class (WMC)-**
  - The WMC for thread based testing class is 4 as it has 4 member functions.
  - Similarly, WMC for cluster based testing is 3 as it has 3 member functions.

**Table 5:** Aggregation between Thread Based Testing and Cluster Based Testing

<table>
<thead>
<tr>
<th>Testing relation and metrics values</th>
<th>WMC</th>
<th>NOA</th>
<th>NOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregation between Thread Based Testing and Cluster Based Testing</td>
<td>4(TBT)</td>
<td>1(TBT)</td>
<td>4(TBT)</td>
</tr>
<tr>
<td></td>
<td>3(CBT)</td>
<td>1(CBT)</td>
<td>3(CBT)</td>
</tr>
</tbody>
</table>

7. Inheritance between partition based, state based, attribute based and category based testing:

- **a) Depth of inheritance tree (DIT) –**
  - Here DIT for integration testing class is 3 as it has 3 ancestor classes.
  - Here DIT values follow our standard (0-4)

- **b) Number of Children (NOC) –**
  - Here NOC value for class Integration is 3.
  - Here NOC values follow our standard (1-4)

- **c) Number of Attributes per Class (NOA)-**
  - For thread, class based, use based and integration testing NOA=1.
  - Here NOA values follow our standard (2-5)

- **d) Number of Methods per Class (NOM)-**
  - For state based NOM=3
  - For attribute based NOM=3
  - Here NOM values follow our standard (3-7)

- **e) Weighted method per class (WMC)-**
  - For state based WMC=3
  - For attribute based WMC=3
  - For category based WMC=4
  - For partition testing WMC=2
  - Here NOM values follow our standard (1-50)

**Table 6:** Aggregation between State Based Testing and Attribute Based Testing

<table>
<thead>
<tr>
<th>Testing relation and metrics values</th>
<th>WMC</th>
<th>NOA</th>
<th>NOM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8. Inheritance between class based, partition based, random based testing:

**Table 7:** Relationship between Partitions Based Testing, Category Based Testing, and Attribute Based Testing

<table>
<thead>
<tr>
<th>Testing relation and metrics values</th>
<th>WMC</th>
<th>NOA</th>
<th>NOM</th>
<th>DIT</th>
<th>NOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relationship between Partition Based Testing</td>
<td>2(PBT)</td>
<td>1(PBT)</td>
<td>2(PBT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Category Based Testing</td>
<td>3(SBT)</td>
<td>1(SBT)</td>
<td>3(SBT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing State Based Testing and Attribute Based Testing</td>
<td>3(ABT)</td>
<td>1(ABT)</td>
<td>3(ABT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class Based Testing</td>
<td>4(CABT)</td>
<td>1(CABT)</td>
<td>1(CABT)</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

For partition testing NOM=2
Here NOM values follow our standard (3-7)

e) Weighted methods per class (WMC)-
For state based WMC=3
For attribute based WMC=3
For category based WMC=4
For partition testing WMC=2
Here WMC values follow our standard (1-50)

**Table 8:** Inheritance between Class Based Testing, Partition Based Testing, and Random Based Testing

<table>
<thead>
<tr>
<th>Testing relation and metrics values</th>
<th>WMC</th>
<th>NOA</th>
<th>NOM</th>
<th>DIT</th>
<th>NOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inheritance between Class Based Testing, Partition Based Testing, and Random Based Testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing State Based Testing</td>
<td>4(CLBT)</td>
<td>1(CLBT)</td>
<td>4(CLBT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Category Based Testing</td>
<td>2(PBT)</td>
<td>1(PBT)</td>
<td>2(PBT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partition Based Testing and Random Based Testing</td>
<td>5(RBT)</td>
<td>2(RBT)</td>
<td>5(RBT)</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

5. RESULT ANALYSIS AND DISCUSSION

Here we summaries our work from above tables for realizing this model through attribute relationship and determine quality of model using different set of metrics, and finally most of the values of our architectural model are follows standard values and decide the value quality of model.

WMC:-The Higher WMC values correlate with increased development, testing and maintenance efforts. As a result of inheritance, the testing and maintenance efforts for the derived classes also increase as a result of higher WMC for a base class (0-50). Above table shows for each component or relation values in between 0 to 50.

DIT:- Inheritance (generalization), is a key concept in the object model. While reuse potential goes up with the number of ancestors, so does design complexity, due to more methods and classes being involved. Studies have found that higher DIT counts correspond with greater error density and lower quality. A class situated too deeply in the inheritance tree will be relatively complex to develop, test and maintain. It is useful, therefore, to know and regulate this depth. A compromise between the high performance power provided by inheritance and the complexity which increases with the depth must be found. A value of between 0 and 4 respects this compromise.

RFC:-Larger RFC counts correlate with increased testing requirements.
LCOM: - A higher LCOM indicates lower cohesion. This correlates with weaker encapsulation, and is an indicator that the class is a candidate for disaggregation into subclasses. This metric measures the correlation between the methods and the local instance variables of a class. High cohesion indicates good class subdivision. Lack of cohesion or low cohesion increases complexity. LCOM range 0 to 1 with zero representing perfect cohesion (each method accesses all attributes), however we have noticed that some values exceed 1.

NOA: - A class with too many attributes may indicate the presence of coincidental cohesion and require further decomposition, in order to better manage the complexity of the model. If there are no attributes, then serious attention must be paid to the semantics of the class, if indeed there are any. A high number of attributes (>10) probably indicates poor design, notably insufficient decomposition. A value of between 2 and 5 respects this compromise.

NOC: - If Values of NOC is larger than reuse of classes also increases, and by this reason increased testing. A class from which several classes inherit is a sensitive class, to which the user must pay great attention. It should, therefore, be limited, notably for reasons of simplicity. A value of between 1 and 4 respects this compromise.

NOV: - would indicate that a class has operations, but not too many. A value greater than 7 may indicate the need for further object-oriented decomposition, or that the class does not have a coherent purpose. This information is useful when identifying a lack of primitiveness in class operations (inhbiting re-use), and in classes which are little more than data types. A value of between 3 and 7 respects this compromise.

CBO: - Excessive coupling limits the availability of a class for reuse, and also results in greater testing and maintenance efforts. Use links between classes define the detailed architecture of the application, just as use links between packages define the highest level architecture. These use links play a determining role in design quality, notably in development and maintenance facilities. Value of 0 indicates that a class has no relationship to any other class in the system, and therefore should not be part of the system. A value between 1 and 4 is good, since it indicates that the class is loosely coupled. A number higher than this may indicate that the class if too tightly coupled with other classes in the model, which would complicate testing and modification, and limit the possibilities of re-use.

6. CONCLUSION

In this paper we implements a set of metrics for measurement of testing model, used to evaluate the quality of the architectural models. Certain model characteristics are measured against quality criteria determined by users thereby allowing you to check that your models meet these quality criteria and appraise the overall quality of a project and find out development of different sub-systems is standard or not. This research work used for developing industrial tool for larger data set, and finally most of the values of our architectural model are follows standard values hence our architecture are useful for any testing process.

REFERENCES


[6]. Jilkes van Gurp, Object Oriented Testing Reports, software verification and validation, DAD404, IDE, University of karlskrona/Ronneby, 1998.


