Chapter 6
CONCLUSIONS AND FUTURE WORK

Research on software metrics devised to measure the quality of object-oriented software has come a long way. Right from the days of traditional metrics to the modern object-oriented metrics, numerous metrics have been evolving. Chapter 2 presents the literature survey about the history of software metrics including the traditional and object-oriented metrics. Out of all such metrics, most researchers have emphasized over the need and applicability of object-oriented metrics that are obtained from the static analysis of object-oriented software. Metrics such as CK metrics [Chidamber and Kemerer(1994)] have been by far the most popular and widely used static object-oriented metrics. These metrics were called static as they were evaluated from the source code or design analysis of a software. Although these metrics ruled the software engineering processes for about 20 years and continue to do so, it has been found that these metrics are not capable of predicting the runtime behavior of an object-oriented software, as they are based on the source code analysis and do not work at all with any runtime information about an object-oriented software. It was understood that the runtime or dynamic information defines the actual behavior of a software in contrast to the expected behavior defined by the source code or static analysis. Mitchell and Power(2003a,2003b,2003c,2004a,2004b&2005) proved this fact in their work on dynamic coupling and cohesion metrics. Working on these lines, many dynamic or runtime metrics have been proposed in the past decade. Yacoub et al.(1999) concluded the dynamic metrics can be used to measure the actual runtime properties of an application as compared to the expected properties measured by static metrics. Most of the dynamic metrics devised till date measure dynamic coupling. A few metrics measure dynamic cohesion and dynamic complexity. Mitchell and Power(2003b) emphasized on the need of dynamic metrics to capture the impact of inheritance on dynamic coupling. There are no metrics proposed till date for measuring the dynamic inheritance or their impact on dynamic coupling.

This research work is primarily directed towards devising such new object-oriented software metrics that can help to estimate the quality of a software on the basis of information collected at runtime. Such metrics known as dynamic metrics are better predictors of the object-oriented software quality than the static metrics that are evaluated from the source code (or bytecode for Java software). All the proposed dynamic metrics
are discussed in Chapter 3 and are listed in Table 29. The new metrics are used to help trace such differences using statistical analysis. Chapter 4 explains the methodology followed and implemented to achieve the research goals. A set of four new dynamic coupling metrics and five dynamic inheritance metrics are proposed, and further validated using Educational Institution Project (EIP). Coupling and inheritance metrics were correlated to see the effect of inheritance on coupling at runtime. A correlation study was also conducted among all the coupling metrics and among all the inheritance metrics. System-level dynamic inheritance metric results were also analyzed to study the impact of inheritance on coupling at runtime. Three more dynamic metrics were proposed. Out of these, one measures the percentage of constructor invocations for a class to know the percentage combined inactivity of the objects of a class at runtime, and the other two measure method-level coupling in a system. All the results are analyzed in Chapter 5.

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Coupling Metrics</strong></td>
</tr>
<tr>
<td>1</td>
<td>DCa (Dynamic Coupling)</td>
</tr>
<tr>
<td>2</td>
<td>DKSC (Dynamic Key Server Class)</td>
</tr>
<tr>
<td>3</td>
<td>DKCC (Dynamic Key Client Class)</td>
</tr>
<tr>
<td>4</td>
<td>DKC (Dynamic Key Class)</td>
</tr>
<tr>
<td></td>
<td><strong>Inheritance Metrics</strong></td>
</tr>
<tr>
<td>5</td>
<td>DPIC (Dynamic Percentage Inheritance Calls)</td>
</tr>
<tr>
<td>6</td>
<td>DMIF (Dynamic Method Inheritance Factor)</td>
</tr>
<tr>
<td>7</td>
<td>DMIF1 (Dynamic Method Inheritance Factor – Variation1)</td>
</tr>
<tr>
<td>8</td>
<td>DMIF2 (Dynamic Method Inheritance Factor – Variation2)</td>
</tr>
<tr>
<td>9</td>
<td>DENOC (Dynamic Effective Number of Children)</td>
</tr>
<tr>
<td></td>
<td><strong>Additional Metrics</strong></td>
</tr>
<tr>
<td>10</td>
<td>DPCCR (Dynamic Percentage Constructor Calls Received)</td>
</tr>
<tr>
<td>11</td>
<td>MQFS (Method Request For Service)</td>
</tr>
<tr>
<td>12</td>
<td>MAM (Most Active Method)</td>
</tr>
</tbody>
</table>

**Table 29: Dynamic object-oriented metric suite**

This chapter further lists the contributions of this research work towards the software engineering community. These are followed by general conclusions and future recommendations.
6.1 Contributions of Thesis

The contributions made by this thesis work are summarized as follows:

- A thorough literature survey and compilation of static and dynamic object-oriented metrics proposed till date.
- Correlation study of static and dynamic metrics
- A new dynamic metric suite for object-oriented software systems consisting of 12 dynamic metrics.
- Validation of the new metric suite.
- Validation of some of the previously proposed metrics such as dynamic CBO.
- Correlation among new metrics.
- Correlation among new and old metrics.
- Correlation among metrics to find the relationship between coupling and inheritance at runtime.
- Theoretically related dynamic coupling and inheritance to the external quality attributes like maintenance, reuse etc.

6.2 Conclusions

- Dynamic metrics demonstrate the actual behavior of a software at runtime in comparison to the static metrics that provide an idea of the actual behavior.
- A dynamic value of a metric can be more than or less than or equal to the static value. If dynamic value is less than or greater than the static value, it is because of one or more of the following factors:
  - **User Input**: It can control the execution of a certain part of source code at runtime.
  - **Inheritance**: A class C appearing to be calling the method of another class D may actually end up accessing one of latter’s super class methods at runtime because of inheritance. Thus, not including class D among the classes to which class C sends a method call and using D’s superclass instead.
  - **Dynamic Binding**
- The information about a software captured by a dynamic metric is strengthened by comparing it to the static metric from which it is derived.
• The differences between the static and dynamic values of a metric increase in case of call-weighed metrics i.e. the metrics than count the number of calls sent or received. This is mainly due to the control loops (e.g. for, while, do-while) being controlled by the user input at runtime.

• It helps in comparing the dynamic and static values of a metric if they are evaluated by taking their percentage over a specific static entity like number of classes in class-level metrics. In this way, dynamic metric also adds to its definition a static information component in the form of that static entity.

• A similar set of dynamic and static values for a metric indicate the following:
  o Less number of control structures used
  o Less execution flow control with user inputs
  o Low degree of inheritance used
  o Low degree of polymorphism used
  o Low degree of dynamic binding used
  All the above mentioned are strong object-oriented concepts and thus would almost always be there in modern object-oriented based applications. Thus the static and dynamic values would rarely be found equal or similar in real world object-oriented applications.

• Dynamic metrics for software applications that are largely based upon user input must be decided over a number of runs and not a single run. An average could be taken of the values found over all the runs. Such a procedure can also extract a useful user input (preferred set of user input) related information.

• Dynamic metrics can be used as improving agents for static metrics by extracting the real time behavior of a software at runtime and taking the appropriate steps (such as eliminating a class that is never used at runtime). After the required changes made in the design phase, value of the static metric can be checked against a threshold value to decide whether there is a further need of dynamic analysis or not.

• Static analysis takes much less cost and effort than dynamic analysis because it is done at early design stage whereas dynamic analysis is performed in the late development stage of software engineering. It makes the previous conclusion even more sensible.

• There is a definite impact of inheritance on coupling at runtime.
6.2 Looking into the future

Based on our research done over dynamic metrics for object-oriented systems, we propose following future work that could be done to enhance the current work:

- To apply the proposed metrics to more real-world object-oriented applications to strengthen their validation.
- To find a way of using the combination of static and dynamic metric values to make the necessary alterations in the design of an application software.
- To explore proposed dynamic inheritance metrics and develop new such metrics.
- To develop such hybrid metrics that combine functions of dynamic coupling and inheritance metrics.
- This study did not find any definite correlation between static and dynamic metrics. Wherever a correlation exists, it seems to be incidental or has more than one specific reason in the given case study. We have listed some of such reasons. But it could be important to track all those reasons for which static and dynamic metrics always correlate. This could only be done by exercising the static/dynamic correlation using many more real-world application software.
- We are working on DynaMetrics tool to be available for commercial use. More such tools need to be developed in order to make more researchers work on these areas.
- We have proposed some additional future dynamic metrics that can measure attributes like dynamic polymorphism etc. Researchers can further explore these metrics and validate them. These metrics are briefly described in Appendix-B.

Chapter Summary

Chapter 6 concludes this thesis work listing all the conclusions drawn from the research work. The important contributions made by this research work to the software engineering community have also been listed. Conclusions and contributions are followed by carefully compiled future recommendations.