Chapter 5

RESULTS AND ANALYSIS

This chapter analyzes the metric results obtained from static and dynamic analysis of 4 EIP object-oriented systems. The test cases have been already explained in the previous chapter. There are 4 test cases each comprising of a set of classes. There are 5, 10, 30, 47 classes in each test case respectively.

5.1 Analysis Techniques

Four test systems of Educational Institution Project (System1, System2, System3 and System4, Refer Table 13), written in Java 5.0, were taken as the sample object-oriented systems for metric validation. For each of the sample programs, the distribution (mean) and variance (standard deviation) of each measure across the classes is calculated. Percentiles (25% and 75%) of data distributions are analyzed for each metric. Normality of data is analyzed using P-P plots. These statistics are used to select metrics that exhibit enough variance to merit further analysis, as a low variance metric would not differentiate classes very well and therefore would not be a useful predictor of external quality. Descriptive statistics will also aid in understanding the results of subsequent analysis.

A correlation study was undertaken to investigate how strongly the metrics are related. The Pearson or product moment correlation test was used. The correlation coefficient $r$ is a number that summarizes the direction and degree of linear relations between two variables and is also known as the Pearson Product-Moment Correlation Coefficient. $r$ can take values between -1 through 0 to +1. When the correlation is positive ($r > 0$), as the value of one variable increases, so does the other. The closer $r$ is to zero the weaker the relationship. If a correlation is negative, when one variable increases, the other variable decreases. The following general categories indicate a quick way of interpreting a calculated $r$ value:

- 0.2: Very weak to negligible correlation
- 0.2 to 0.4: Weak, low correlation (not very significant).
- 0.4 to 0.7: Moderate correlation.
- 0.7 to 0.9: Strong, high correlation
- 0.9 to 1.0: Very strong correlation
Any relationship between two variables should be assessed for its *significance* as well as its strength. A standard two tailed t-test was used to determine whether the correlation coefficient was statistically significant. Coefficients were considered significant if the t-test p-value was below 0.05. This tells how unlikely a given correlation coefficient, $r$, will occur given no relationship in the population. Therefore, smaller the p-level, more significant the relationship is.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Number of classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>System1</td>
<td>5</td>
</tr>
<tr>
<td>System2</td>
<td>10</td>
</tr>
<tr>
<td>System3</td>
<td>30</td>
</tr>
<tr>
<td>System4</td>
<td>47</td>
</tr>
</tbody>
</table>

**Table 13: Education Institution Project – Test case size**

We have divided the result analysis in three major categories:

- Coupling metrics
- Inheritance metrics
- Coupling – Inheritance metric correlation
- Additional Metrics
  - Constructor-based automatic initialization metrics
  - Method-level coupling metrics

### 5.2 Coupling Metrics Results

Following are the class-level dynamic coupling metrics from the new suite (except DCBO [Mitchell and Power(2004a)]):

- Dynamic Coupling between Objects (DCBO)
- Dynamic Afferent Coupling (DCa)
- Dynamic Key Server Class (DKSC)
- Dynamic Key Client Class (DKCC)
- Dynamic Key Class (DKC)

#### 5.2.1 Descriptive Analysis

A descriptive study for the test case (Educational Institution Project - System1-4) data was conducted using mean, standard deviation, percentiles for both the static as well as
dynamic analysis values collected for each coupling metric. All static and dynamic measures (for all the 4 systems used in the validation) exhibited sufficient variances which make them suitable candidates for further analysis. But these variances differ as we go from static to dynamic values. Further statistical analysis techniques required a normal data distribution. P-P plots showed a normal distribution for all the four systems. Any data that did not demonstrate a normal distribution using P-P plots was transformed computing the logarithm of each data point. Next we do a system-wise detailed analysis of test case data for all the coupling measures.

5.2.1.1 System-wise Metric Data Analysis

This subsection analyzes the static and dynamic metric data for each test case EIP system. We have divided this subsection into four parts namely System1, System2, System3 and System4. For each system, static and dynamic analysis for each of the new coupling metrics is discussed.

a) System 01

Comparing the means of static and dynamic metric values, there are clear variations in means for KSC and KC metrics from static to dynamic, whereas there are no variations in the other 3 metrics. This shows that the difference in static and dynamic values for KC coupling metric is mainly due to KSC that is based on calls received by the classes, rather than KCC, that is based on calls sent out to other classes. The results of coupling metrics based on data given in Figure 14 are discussed below:

- The descriptive statistical data, for both the static and dynamic CBO, comes out to be the same showing no differences between static and dynamic analysis results.
Class *LibraryManager* has maximum values (60% each) for both SCBO and DCBO. This means that *LibraryManager* sends calls to 60% of the total number of classes both before runtime and at runtime. The high CBO value for the class would make it less maintainable, less reusable, less portable, with low testability.

- It is evident that the distribution of Ca metric values among the classes has changed at runtime. For Ca, 25% of classes have a value less than 10.00 whereas at runtime this value goes down to .00 showing a reduction in the metric values at runtime. But the scenario is different for the top 25% of the classes as they go beyond the 30.00 value whereas for static this number is 20.00. So overall dynamic values seem to be more distributed than the static values for Ca that is also evident from a higher standard deviation value for dynamic Ca. However, the static and dynamic means remain the same which is incidental. For Static Ca, four of the five classes have the same values (20% each). These classes are *BookImpl, EmployeeInfoImpl, IssueInfoImpl, LibraryImpl*. These static values vary at runtime and class *BookImpl* give the maximum value for Dynamic Ca. This value is 40%. Thus the number of classes from which *BookImpl* receives calls are doubled at runtime. Similarly for class *EmployeeInfoImpl* this count goes from 20% for static Ca to 0% at runtime. Thus no class is accessing its methods at runtime. Thus there is no clear strongest coupled class before runtime whereas class *BookImpl* comes out to be the most strongly coupled at runtime. There is a more balanced afferent coupling among classes before runtime than at runtime. The high Ca value for *BookImpl* class would make it less maintainable, more reusable, less portable, less testable, whereas these external attributes would have an average moderate values for the classes before runtime too because of a similar afferent coupling.

- Values for Mean, standard deviation and percentiles all decrease at runtime indicating that on average lesser number of methods calls are actually being sent than expected at runtime. Class *BookImpl* is the Key Server Class (KSC) both before and at runtime. Its SKSC percentage count is 45.4% and DKSC percentage count is 18.2%. Thus there is a clear decline of about in the value at runtime. DKSC is approx. 40% of SKSC which is a huge variation for a primary server class that is sending the maximum amount of messages to other classes. The high KSC value for *BookImpl* class would make it less maintainable, more reusable, less portable and less testable.
Table 15: PAC (system) and DCBO (system) data for all four systems

<table>
<thead>
<tr>
<th>Test Case</th>
<th>PAC (%)</th>
<th>DCBO - System (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System1</td>
<td>80.00</td>
<td>64.71</td>
</tr>
<tr>
<td>System2</td>
<td>100.00</td>
<td>70.58</td>
</tr>
<tr>
<td>System3</td>
<td>73.33</td>
<td>65.68</td>
</tr>
<tr>
<td>System4</td>
<td>53.19</td>
<td>71.17</td>
</tr>
</tbody>
</table>

- KCC values have same means, standard variations and percentiles for static and dynamic evaluation. KCC means are on the higher side as compared to the other metrics but the standard deviation is quite low indicating a less distributed data. It can be noted that the about 25% of the classes have a value more than 50, which indicates a higher usage of these classes. Class `LibraryManager` is the Key Client Class (KCC) both before and at runtime. Both SKCC and DKCC percentage counts are 90.9%. Thus there is a no variation in value at runtime. This means all the method calls that were expected to be executed at runtime were executed as

![Normal P-P Plot of Static CBO](image1.png) ![Normal P-P Plot of Dynamic CBO](image2.png)

**Figure 19:** P-P plots of Static and Dynamic CBO for System1

![Normal P-P Plot of Static Ca](image3.png) ![Normal P-P Plot of Dynamic Ca](image4.png)

**Figure 20:** P-P plots of Static and Dynamic Ca for System1
Design and Validation of Dynamic Metrics for Object-Oriented Software Systems

Figure 21: P-P plots of Static and Dynamic KSC for System1

Figure 22: P-P Plots of Static and Dynamic KCC for System1

Figure 23: P-P plots of Static and Dynamic KC for System1
suggested by static analysis. Thus class LibraryManager would have a very low maintainability, reusability, portability and testability.

- KC values are here seem to be more effected by KSC than KCC as their means and percentiles decline from static to dynamic analysis much in the same manner as KSC than KCC. Class LibraryManager is the Key Class (KC) both before and at runtime. In other words, LibraryManager is the most coupled class (call-weighed). Both of SKC and DKC percentage counts are 45.4%. Thus there is a no variation in value at runtime. This means all the method calls that were expected to be executed at runtime were executed as suggested by static analysis. This is also due to the fact that KC adds up the number of calls received counted by KSC and number of calls sent out counted by KCC. Thus the sum of number of calls received and number of calls sent out must be same for both static and dynamic analysis. Thus class LibraryManager would have a very low maintainability, reusability, portability and testability. It is important to note here that reusability factor increases with an increase of KSC count whereas it decreases with an increase of KCC count. Thus reusability must be a factor of a positive or a negative difference between the KCC and KSC that combine to evaluate the respective KC value. Thus a class having a high KC value can result in a high or low reusability depending upon the positive or negative differences between its KCC and KSC values. KC value of LibraryManager class is comprised 100% of KCC value both before runtime and at runtime. Thus inspite of having a high KC value, LibraryManager class has low reusability because of a huge positive difference between its KCC (90%) and KSC (0%) values.

Figure 24: Dynamic CBO – System1
From Table 15, DCBO for System1 is 64.71% evaluated from data collected for all the class at runtime. This value indicates that out of the total accesses in the system at runtime, 64.71% are sent outside the source classes (i.e. Inter-Class Coupling), hence contributing to inter-class coupling of classes excluding any intra-class cohesion (method-method) calls for the classes.

PAC is a system-level metric derived from DKC metric that works at class granularity. PAC for the system1 comes out to be 80% which is a very healthy percentage. This shows that 80% of the total number of classes is coupled to other classes (incoming or outgoing coupling) at runtime.

b) System02

Comparing the means of static and dynamic metric values from Table 16, there is a clear deviation for all the metrics from static to dynamic. The results of coupling metrics based on data given in Figure 16 are discussed below:

- Both SCBO and DCBO have high means, standard deviations and percentile values but there is a clear collective rise in all these values at runtime. The dynamic values seem to be more distributed with a higher variance than static values. Class Institution1 has maximum values (60% and 70% respectively) for both SCBO and DCBO. This means that Institution1 sends calls to 60% of the total number of classes before runtime and this count even rises by 10% at runtime to give a DCBO of 70%. The high CBO value for the class would make it less maintainable, less reusable, less portable and less testable. It is also noticeable that all these external quality attributes even suffer a negative impact of the rise of CBO from static to dynamic analysis.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Size</th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>Percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Static</td>
<td>Dynamic</td>
<td>Static</td>
</tr>
<tr>
<td>CBO</td>
<td>10</td>
<td>13.00</td>
<td>16.00</td>
<td>18.89</td>
</tr>
<tr>
<td>Ca</td>
<td>10</td>
<td>13.00</td>
<td>16.00</td>
<td>11.59</td>
</tr>
<tr>
<td>KSC</td>
<td>10</td>
<td>10.00</td>
<td>5.93</td>
<td>9.410</td>
</tr>
<tr>
<td>KCC</td>
<td>10</td>
<td>10.00</td>
<td>17.77</td>
<td>21.52</td>
</tr>
<tr>
<td>KC</td>
<td>10</td>
<td>10.00</td>
<td>11.85</td>
<td>9.80</td>
</tr>
</tbody>
</table>

Table 16: Descriptive analysis – System2
• Dynamic Ca values have higher means and standard deviations than the static Ca values indicating higher incoming coupling at runtime. For Static Ca, class BookImpl has the maximum value of 40%. BookImpl also has the maximum value for Dynamic Ca making it the class with maximum incoming coupling both before and during runtime. Ca rises to 50% at runtime. Thus the number of classes from which BookImpl receives calls are increased by 10% at runtime. The high Ca value for the class would make it less maintainable, more reusable, less portable and tough to test. Also it can be seen that all these external attributes are even lesser for the class at runtime in comparison to the results of static analysis. Similarly for class EmployeeInfoImpl this count goes from 0% from static Ca to 20% at runtime. Thus its actual incoming coupling (that showed 0% statically) rises to 20% at runtime.

Figure 25: P-P plots of Static and Dynamic CBO for System2

Figure 26: P-P Plots of Static and Dynamic Ca for System2
Design and Validation of Dynamic Metrics for Object-Oriented Software Systems

Figure 27: P-P plots of Static and Dynamic KSC for System2

Figure 28: P-P plots of Static and Dynamic KCC for System2

Figure 29: P-P Plots of Static and Dynamic KC for System2
• KSC shows good enough means, variations and percentile distributions both before runtime and at runtime. But KSC values dip as we go from static to dynamic analysis results, hence showing a lesser incoming calls received by server classes at runtime than expected before runtime. Class BookImpl is the Key Server Class (KSC) both before and at runtime. Its SKSC percentage count is 29.6% and DKSC percentage count is 18.5% which is less than SKSC. This shows that BookImpl receives lesser number of calls at runtime (actual number of calls received) i.e. its demand by other classes decreases at runtime. It can also be seen that both of its static and dynamic KSC values are much higher than that of all the other classes (except class LibraryImpl that has a high SKSC value, but its DKSC value is still quite low). Thus class BookImpl possesses the strongest incoming coupling (call-weighed) in the system both before and during runtime. The high KSC value for the class would make it less maintainable, more reusable, less portable and tough to test. Also it can be seen that all these external attributes are lesser for the class at runtime in comparison to the results of static analysis.

• KCC shows the highest dynamic mean and standard deviation for System2 among all the 5 metrics. This inturn leads to a higher 75% value as well. There is a steep rise from static to dynamic descriptive values indicating a higher number of calls being sent at runtime. Class Institution1 is the Key Client Class (KCC) both before and at runtime. SKCC and DKCC values are 70.4%. Thus class Institution1 sends the most number of calls to other classes both before and during runtime. There is a large variation for class BookImpl from 0% before runtime to 48.1% during runtime. Similarly there is a substantial rise in metric values for classes IssueInfoImpl and MegazineImpl. The high KCC value for the class would make it less maintainable, less reusable, less portable and tough to test.

• Although means for KC show more impact of KCC than KSC but standard deviation indicates otherwise. KC has a large fall in variance at runtime. Class Institution1 is the Key Class (KC) both before and at runtime with each analysis showing a value of 35.2%. In other words, Institution1 is the most coupled class (call-weighed). Thus there is a no variation in value at runtime. This means all the method calls that were expected to be executed at runtime were executed as suggested by static analysis. This is also due to the fact that KC adds up the number of calls received counted by KSC and number of calls sent out counted by KCC.
The higher KCC value than KSC (in KC) for the class would make it less maintainable, less reusable, less portable and tough to test. Thus the sum of number of calls received and number of calls sent out must be same for both static and dynamic analysis. DKC value for class BookImpl is quite close to that of Institution1 at 33.3% whereas SKC value is quite lower than that of SKC value for Institution1. There is a decline in the KC value for the classes Clerk, LibraryImpl and NewspaperImpl.

- DCBO (System) is 70.58% evaluated from data collected for all the class at runtime. This value indicates that out of the total accesses in the system at runtime, 70.58% are sent outside the source classes, hence contributing to high coupling of classes excluding cohesion of a class.

- Percentage Active Classes (PAC) for the system comes out to be 100% which is a very healthy percentage. This shows that all the classes are active at runtime and are coupled to other classes (incoming or outgoing coupling) at runtime.

c) System03
All the descriptive statistical values are low for System4. Comparing the means of static and dynamic metric values, there is a clear deviation for all the metrics from static to dynamic. The results of coupling metrics based on data given in Figure 17 are discussed below:

- The mean for CBO slightly decreases at runtime whereas the standard deviation increases at runtime. Percentiles show the overall lower metric values. Class Institution2 has the maximum values (23.3% and 36.7% respectively) for Static as well as Dynamic
Chapter 5 – Results and Analysis

Table 17: Descriptive analysis – System3

<table>
<thead>
<tr>
<th>Metric</th>
<th>Size</th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>Percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Static</td>
<td>Dynamic</td>
</tr>
<tr>
<td>CBO</td>
<td>30</td>
<td>4.67</td>
<td>3.78</td>
<td>5.37</td>
</tr>
<tr>
<td>Ca</td>
<td>30</td>
<td>4.56</td>
<td>3.78</td>
<td>5.430</td>
</tr>
<tr>
<td>KSC</td>
<td>30</td>
<td>3.33</td>
<td>1.83</td>
<td>3.972</td>
</tr>
<tr>
<td>KCC</td>
<td>30</td>
<td>3.33</td>
<td>6.61</td>
<td>5.67</td>
</tr>
<tr>
<td>KC</td>
<td>30</td>
<td>3.33</td>
<td>4.24</td>
<td>3.28</td>
</tr>
</tbody>
</table>

CBO. Thus CBO even increases at runtime. There is a considerable variation for rest of the classes as well. The high DCBO value for the class would make it less maintainable, less reusable, less portable and tough to test. As DCBO value is higher than SCBO, it indicates that the external attributes are even less than those expected before runtime.

- Ca means and standard deviations are quite similar to that of CBO. There is no substantial variation static and dynamic descriptive data. Class Result has the highest static Ca (incoming coupling) value of 26.7%. DCa (13.3%) decreases for Result class considerably at runtime. At runtime, class StudentAthlete has the highest DCa value of 16.7% whose static Ca value was 10%. This means that the methods of StudentAthlete class are accessed by highest number of classes in the system at runtime. The high Ca value for the class would make it less maintainable, more reusable, less portable and tough to test. Thus while class Result has the lowest maintainability, lowest portability, lowest testability and highest reusability in the system before runtime, the runtime results show a completely different picture. Here, the maintainability, portability and testability increase whereas reusability decreases for class Result as the Ca value falls at runtime. Similarly for class StudentAthlete maintainability, portability and testability decrease whereas reusability increases as we go from static to dynamic metric values.

- Dynamic means, variances and 75% percentile values drop as compared to static descriptive values which indicates lesser amount of incoming calls executed at runtime, leading to lower dynamic coupling. Classes BookImpl and Result both are Static Key Server Classes (SKSC) and Class StudentAthlete is the Dynamic Key Server Class (DKSC). SKSC value for BookImpl and Result is 12.9% and DKSC for StudentAthlete is 8.1%. Both of these values are quite low. Thus the total
number of calls received by all the classes are well distributed among the classes. Static and Dynamic values for almost all the classes vary as shown by their respective SKSC and DKSC values. *BookImpl* and *Result* classes both are less maintainable, less portable,

Figure 31: P-P plots of Static and Dynamic CBO for System3

Figure 32: P-P plots of Static and Dynamic Ca for System3

Figure 33: P-P plots of Static and Dynamic KSC for System3
more reusable and less testable than all the other classes at runtime whereas class StudentAthlete is the least maintainable, least portable, least testable and most reusable class at runtime. A KSC class would also be the busiest class in the system as it spends most of its time receiving calls from other class and it would be even busier if its KCC count is higher as well.

- Noticeably the standard deviation drops at runtime indicating at a lower variance. There is an overall rise in the values at runtime as indicative by the static and dynamic means and percentiles. Class Institution2 is the Key Client Class (KCC) both before and at runtime with 30.2% method calls sent before runtime and 46.03% during runtime. BookImpl seemed to have a small percentage (4.8%) of the total method calls sent out by all the classes before runtime. This percentage rises
considerably at runtime to give a value of 39.7%. Class Institution2 is less maintainable, less portable, less reusable and less testable than all the other classes.

- KC descriptive data shows more prominent effect of KCC than KSC indicating that there are higher percentages of calls sent than received at runtime, contributing to the total number of associations with other classes. Class *Institution2* is the Key Client Class (KC) both before and at runtime. SKC value is 15.2% and DKC value is 23.2%. Thus *Institution2* is the most coupled class (call-weighted). Class *BookImpl* is also quite close to becoming the Key Class at runtime with a DKC value of 22.4% whereas SKC value is quite lower than that of SKC value for Institution1. Class *LibraryCard* is also quite active at runtime having a DKC value of 19.2%. There is a considerable amount of variation from static to dynamic values for almost all the other classes of the application. Class *Institution2* is less maintainable, less portable, less reusable and less testable than all the other classes as its KSC count is 0% both statically and dynamically.

- DCBO (System) for System2 is 65.68% which is quite close to that of System1. This value indicates that out of the total accesses in the system at runtime, 65.68% are sent outside the source classes, hence contributing to high coupling of classes excluding cohesion of a class.

- Percentage Active Classes (PAC) for the system comes out to be 73.33% which is a very healthy percentage. This shows that 73.33% of the total classes are active and are coupled to other classes (incoming or outgoing coupling) at runtime.

![Pie Chart: Dynamic CBO – System3](image-url)

**Figure 36: Dynamic CBO – System3**
d) System04

Comparing the means of static and dynamic metric values, there is a clear deviation for all the metrics from static to dynamic. We see a fall in descriptive data with increase in size of the system indicating an underuse of system classes both before and at runtime. The results of coupling metrics based on data given in Figure 18 are discussed below:

- Static and dynamic descriptive analysis values are low. The dynamic metric means, standard deviations and percentiles are lower than the static metric values indicating an overall drop in coupling at runtime. Class Institution3 has the maximum value for static (21.3%) as well as dynamic (27.7%) CBO. Thus Institution3 sends method calls to most number of classes before runtime and at runtime. Other classes too

<table>
<thead>
<tr>
<th>Metric</th>
<th>Size</th>
<th>Mean Static</th>
<th>Mean Dynamic</th>
<th>Standard Deviation Static</th>
<th>Standard Deviation Dynamic</th>
<th>Percentile 25% Static</th>
<th>Percentile 25% Dynamic</th>
<th>Percentile 75% Static</th>
<th>Percentile 75% Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBO</td>
<td>47</td>
<td>2.63</td>
<td>1.81</td>
<td>3.66</td>
<td>4.41</td>
<td>.00</td>
<td>.00</td>
<td>4.26</td>
<td>2.13</td>
</tr>
<tr>
<td>Ca</td>
<td>47</td>
<td>2.58</td>
<td>1.81</td>
<td>4.254</td>
<td>2.429</td>
<td>.00</td>
<td>.00</td>
<td>4.26</td>
<td>2.13</td>
</tr>
<tr>
<td>KSC</td>
<td>47</td>
<td>2.13</td>
<td>1.04</td>
<td>3.241</td>
<td>1.392</td>
<td>.00</td>
<td>.00</td>
<td>2.44</td>
<td>1.22</td>
</tr>
<tr>
<td>KCC</td>
<td>47</td>
<td>2.13</td>
<td>4.05</td>
<td>4.68</td>
<td>9.76</td>
<td>.00</td>
<td>.00</td>
<td>2.41</td>
<td>2.41</td>
</tr>
<tr>
<td>KC</td>
<td>47</td>
<td>2.13</td>
<td>2.55</td>
<td>2.85</td>
<td>5.13</td>
<td>.61</td>
<td>.00</td>
<td>2.42</td>
<td>2.42</td>
</tr>
</tbody>
</table>

Table 18: Descriptive analysis – System4

show variable results for static and dynamic CBO values. Class Institution3 is less maintainable, less portable, less reusable and less testable than all the other classes both before and at runtime.

- Over coupling seems to be lower at runtime as indicated by the descriptive statistics. This means a lesser number of classes contribute to incoming coupling at runtime. Class Result receives calls from highest number of unique classes both before runtime (SCa value 19.1%) and during runtime (DCa value 10.6%). Class GeneralOperations also shows a high SCa value of 10.6% but it reduces considerably at runtime (4.3%). Other classes also show variable results for static and dynamic Ca values. Class Result is less maintainable, less portable, more reusable and less testable than all the other classes both before and at runtime. It can be observed that there is a large variation in Ca from static to dynamic metric values. This means that the class will be more maintainable, more portable, less reusable and more testable at runtime than expected from the static analysis.
Design and Validation of Dynamic Metrics for Object-Oriented Software Systems

Figure 37: P-P plots of Static and Dynamic CBO for System4

Figure 38: P-P plots of Static and Dynamic Ca for System4

Figure 39: P-P plots of Static and Dynamic KSC for System4
The call-weighed metric KSC also similar descriptive stats as Ca metric and there is again a drop from static and dynamic descriptive statistical values. Class *Student* is the Static Key Server Class (SKSC) and class *Result* is the Dynamic Key Server Class (DKSC). Class *Student* has a SKSC value of 12.1% whereas class *Result* has a DKSC value of 6.1%. *Result* has also got a strong SKSC value of 10.9%. Class *Student’s* KSC value reduces considerably from 12.1% before runtime to 1.21% at runtime showing a drop in incoming call-weighed coupling at runtime. Class *Student* is less maintainable, less portable, more reusable and less testable than all the other classes before runtime whereas it is overtaken by class *Result* at runtime.

- KCC has high means and standard deviations for both the static and dynamic values but most noticeable fact is a large rise in the metrics means and standard deviations at runtime. This indicates a large increase in the calls sent at runtime. Class *Institution3* is the Static Key Client Class (SKCC) and class *BookImpl* is the
Dynamic Key Client Class (DKCC). Class \textit{Institution3} has a SKCC value of 31.3%. This value rises to 42.2% at runtime (DKCC value) but is not enough to make it the Key Client Class at runtime. This value is less than the dynamic value of actual Key Client Class \textit{BookImpl} whose DKCC value is 44.6%. There is a large variation in static and dynamic values of Class \textit{MegazineImpl} also. Thus showing that a class that is not expected to be active (and coupled to other classes) at all before runtime actually can show substantial coupling at runtime. Class \textit{Institution3} is less maintainable, less portable, less reusable and less testable than all the other classes expected before runtime whereas class \textit{BookImpl} has the least maintainability, least portability, least reusability and least testability as shown by the dynamic results.

- KC seems to have more impact of KCC than KSC as its descriptive statistics are similar to that of KCC. This goes to show that the class depends more upon calls sent for exhibiting coupling at runtime than the received calls. Class \textit{Institution3} is the Static Key Class (SKC) and Class \textit{BookImpl} is the Dynamic Key Class (DKC). Thus \textit{Institution2} is the most coupled class (call-weighed) statically having a SKC value of 15.6%. Class \textit{BookImpl} is the actual coupled class evaluated from dynamic analysis having a dynamic KC count of 24.2%. Class \textit{Institution3} is also quite close to becoming the key class at runtime with a DKC value of 21.2%. On the other hand SKC value (7.2%) for \textit{BookImpl} is quite lower than that of SKC value for \textit{Institution3}. Class \textit{Institution3} is less maintainable, less portable, less reusable and less testable than all the other classes expected before runtime whereas class \textit{BookImpl} is least maintainable, least portable, least reusable and least testable at runtime.

![Figure 42: Dynamic CBO – System4](image)
• DCBO (System) is 71.17% evaluated from data collected for all the class at runtime. This value is quite close to the value for System2 thus showing the runtime behavioral similarity.

• Percentage Active Classes (PAC) for the system comes out to be 53.19% which is a very healthy percentage. This shows that 53.19% of the total classes are active and are coupled to other classes (incoming or outgoing coupling) at runtime.

5.2.2 Correlation Study

5.2.2.1 Static/Dynamic Correlation

It was important to conduct a correlation study to check whether there exists any correlation between the static and dynamic values for a metric. We conducted such studies for all the 5 metrics including the CBO (static and dynamic) metrics over all the four test cases. We also computed the percentage of dynamic value to static value in order to track the percentage variation in the metric value at runtime. A 100% value shows no variation in the dynamic value. NA indicates a zero value for both static and dynamic analysis. Infinity (Inf) indicates a zero static value and a non-zero dynamic value.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Corr.</th>
<th>SCBO /DCBO</th>
<th>SCa /DCa</th>
<th>SKSC /DKSC</th>
<th>SKCC /DKCC</th>
<th>SKC /DKC</th>
</tr>
</thead>
<tbody>
<tr>
<td>System1</td>
<td>p-value</td>
<td>.853</td>
<td>.535</td>
<td>.963</td>
<td>.987</td>
<td>.908</td>
</tr>
<tr>
<td></td>
<td>Sig.</td>
<td>.066</td>
<td>.353</td>
<td>.008</td>
<td>.002</td>
<td>.033</td>
</tr>
<tr>
<td>System2</td>
<td>p-value</td>
<td>.977</td>
<td>.866</td>
<td>.771</td>
<td>.767</td>
<td>.816</td>
</tr>
<tr>
<td></td>
<td>Sig.</td>
<td>.000</td>
<td>.001</td>
<td>.009</td>
<td>.010</td>
<td>.004</td>
</tr>
<tr>
<td>System3</td>
<td>p-value</td>
<td>.879</td>
<td>.666</td>
<td>.672</td>
<td>.605</td>
<td>.712</td>
</tr>
<tr>
<td></td>
<td>Sig.</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
</tr>
<tr>
<td>System4</td>
<td>p-value</td>
<td>.904</td>
<td>.671</td>
<td>.704</td>
<td>.631</td>
<td>.742</td>
</tr>
<tr>
<td></td>
<td>Sig.</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
</tr>
</tbody>
</table>

Table 19: Correlation study – Static vs Dynamic coupling metrics

a) Static/Dynamic CBO

It is seen from the Pearson Correlation study results shown in Table 19 that the static and dynamic CBO are very strongly correlated to each other. This strong correlation indicates that dynamic values increase with increasing static values. It also indicates lesser use of inheritance and dynamic binding. It also shows that the application is not very much dependent upon user input.
Tables 30-33 (Appendix-A) shows the percentage variation in the dynamic values for all the classes of all 4 sample applications. Static and dynamic values for all the four test case systems are also compared using graphs shown in Figures 43-46 and DynaMetrics output snapshots shown in Figures 75-78 in Appendix-A. It can also be seen from the results that for most classes, the metric values decrease or remain same from static to dynamic. This is due to the fact that any dynamic metric that is based on number of unique classes or methods can only exceed its static metric value calculated before runtime (i.e. DCBO>=SCBO) if the class makes use of the dynamic features like inheritance or dynamic binding.

Figure 43: SCBO/DCBO - System1
a) Bar Graph

b) Line Graph

Figure 44: SCBO/DCBO – System2
Design and Validation of Dynamic Metrics for Object-Oriented Software Systems

Figure 45: SCBO/DCBO – System3

- 112 -
a) Bar Graph

b) Line Graph

Figure 46: SCBO/DCBO – System4
b) Static/Dynamic Afferent Coupling

There exists a very strong correlation between the two metrics except for the test case 1 (System1) as shown in Table 19. This indicates that in the case of test case 2, 3 and 4, the number of couples (outgoing) made by a class counted before runtime are directly proportional to those counted at runtime.

Tables 34-37 (Appendix-A) show the percentage variation in the dynamic values for all the classes of all 4 sample applications. Static and dynamic values for all the four test case systems are also compared using graphs shown in Figures 47-50 and DynaMetrics output snapshots shown in Figures 79-82 in Appendix-A. It can also be seen from the results that for most classes, the metric values decrease or remain same from static to dynamic. This is due to the fact that any dynamic metric that is based on number of unique classes or methods can only exceed its static metric value calculated before runtime (i.e. DCa >= SCa) if the class makes use of the dynamic features like inheritance or dynamic binding.

![Bar Graph](image1.png)

**a) Bar Graph**

![Line Graph](image2.png)

**b) Line Graph**

**Figure 47: SCa/DCa - System1**
a) Bar Graph

b) Line Graph

Figure 48: SCa/DCa – System2
Design and Validation of Dynamic Metrics for Object-Oriented Software Systems

Figure 49: SCa/DCa – System3
Figure 50: SCA/DCa – System4

a) Bar Graph

b) Line Graph
c) Static/Dynamic Key Server Class

There exists a very strong correlation, as shown in Table 19, which indicates that the results did not change a great deal at runtime. It also indicates that whenever SKSC increases DKSC also increases for all the 4 test cases.

Tables 38-41 (Appendix-A) show the percentage variation in the dynamic values for all the classes of all 4 sample applications. Static and dynamic values for all the four test case systems are also compared using graphs shown in Figures 51-54 and DynaMetrics output snapshots shown in Figures 83-86 in Appendix-A. It can also be seen from the results that although there are a few exceptions but for most classes, the metric values decrease or remain same from static to dynamic. The reason behind the reduction in metric value from static to dynamic analysis results are the method calls that are counted before runtime but are not actually executed at runtime because of the behavioral changes in the application seen at runtime. These behavioral changes could be because of the dependency of a code area on the user input, the execution of certain loop or the dynamic factors like inheritance or dynamic binding.

![Bar Graph](image1)

**a) Bar Graph**

![Line Graph](image2)

**b) Line Graph**

*Figure 51: SKSC/DKSC – System1*
Chapter 5 – Results and Analysis

a) Bar Graph

b) Line Graph

Figure 52: SKSC/DKSC – System2
a) Bar Graph

b) Line Graph

Figure 53: SKSC/DKSC – System3
Figure 54: SKSC/DKSC – System4

a) Bar Graph

b) Line Graph
d) Static/Dynamic Key Client Class

There is a very strong correlation for all the four test cases as shown in Table 19. This could be due to similar results obtained from static and dynamic analysis. This also means that whenever SKCC increases, DKCC also increases.

Tables 42-45 (Appendix-A) show the percentage variation in the dynamic values for all the classes of all 4 sample applications. Static and dynamic values for all the four test case systems are also compared using graphs shown in Figures 55-58 and *DynaMetrics* output snapshots shown in Figures 87-90 in Appendix-A. It can also be seen from the results that although there are a few exceptions but for most classes, the metric values decrease or remain same from static to dynamic. The reason behind the reduction in metric value from static to dynamic analysis results are the method calls that are counted before runtime but are not actually executed at runtime because of the behavioral changes in the application seen at runtime. These behavioral changes could be because of the dependency of a code area on the user input, the execution of certain loop or the dynamic factors like inheritance or dynamic binding.

![Bar Graph](image1)

**a) Bar Graph**

![Line Graph](image2)

**b) Line Graph**

*Figure 55: SKCC/DKCC – System1*
a) Bar Graph

b) Line Graph

Figure 56: SKCC/DKCC – System2
Design and Validation of Dynamic Metrics for Object-Oriented Software Systems

Figure 57: SKCC/DKCC – System3
Chapter 5 – Results and Analysis

a) Bar Graph

b) Line Graph

Figure 58: SKCC/DKCC – System4
e) Static/Dynamic Key Class

There exists a very strong correlation for all the four test cases as shown in Table 19. This could be due to similar results obtained from static and dynamic analysis. This also means that whenever SKC increases, DKC also increases.

Tables 46-49 (Appendix-A), show the percentage variation in the dynamic values for all the classes of all four sample applications. Static and dynamic values for all the four test case systems are also compared using graphs shown in Figures 59-62 and DynaMetrics output snapshots shown in Figures 91-94 in Appendix-A. It can also be seen from the results that although there are a few exceptions but for most classes, the metric values decrease or remain same from static to dynamic. The reason behind the reduction in metric value from static to dynamic analysis results are the method calls that are counted before runtime but are not actually executed at runtime because of the behavioral changes in the application seen at runtime. These behavioral changes could be because of the dependency of a code area on the user input, the execution of certain loop or the dynamic factors like inheritance or dynamic binding.

![Bar Graph](image1.png)

**a) Bar Graph**

![Line Graph](image2.png)

**b) Line Graph**

**Figure 59: SKC/DKC – System1**
a) Bar Graph

b) Line Graph

Figure 60: SKC/DKC – System2
Design and Validation of Dynamic Metrics for Object-Oriented Software Systems

Figure 61: SKC/DKC – System3

(a) Bar Graph

(b) Line Graph
a) Bar Graph

b) Line Graph

Figure 62: SKC/DKC – System4
5.2.2.2 Inter-metric Correlation

5.2.2.2.1 Correlation with DCBO

We next studied the correlations, if any, between each of the four new dynamic metrics and the Mitchell and Power(2004a) metric DCBO.

a) Dynamic CBO/Dynamic Ca

There is a moderate to very weak correlation (.504, .606, .123, .091) between the two as DCa is based on calls received by a class whereas Dynamic CBO is based on the calls sent out by the class at runtime. Also this dynamic correlation varies from its static counterpart (SCBO/SCa correlation). The correlation (negative) weakens (from static to dynamic) in case of test cases System1 whereas there are no considerable changes for the rest of the test cases. It can also be seen from the results that for most classes, the metric values decreases or remain same from static to dynamic. This is due to the fact that any dynamic metric that is based on number of unique classes or methods can only exceed its static metric value calculated before runtime (i.e. SCBO>=DCBO and SCa>=DCa), if the class makes use of the dynamic features like inheritance or dynamic binding.

b) Dynamic CBO/Dynamic KSC

There is a moderate to weak correlation (.554, .683, .243, .156) between the two as DKSC is based on calls received by a class whereas Dynamic CBO is based on the calls sent out by the class at runtime. Also this dynamic correlation varies from its static counterpart. Their differences go further as DKSC is a call-weighed metric and Dynamic CBO is a class-count metric. The stronger correlation in case of test case System2 is incidental. There are no considerable differences in dynamic correlation results from the static correlation (SCBO/SKSC) results.

c) Dynamic CBO/Dynamic KCC

There is a moderate to strong correlation (.971, .707, .625, .689) between the two metrics as both the metrics are evaluated on the basis of method calls sent out by a class at runtime. Dynamic CBO counts the number of classes to which a class sends messages to whereas DKCC uses the number of calls/messages sent out by a class. A strong correlation indicates that whenever the class count increases in Dynamic CBO, there is an increase in the number of calls in DKCC as well. The correlation seems slightly stronger as compared
to the static correlation (SCBO/SKCC) for System2 where the correlation remains quite same for the rest of the test cases at runtime.

d) Dynamic CBO/Dynamic KC

There is a moderate to strong correlation (.959,.559,.589,.671) because of higher client classes’ contribution than server classes’ contribution in DKC as CBO is highly correlated to KCC. Comparing to the static correlation (SCBO/SCC), the correlation remains quite the same.

5.2.2.2.2 Additional Correlation Study

It was further important to conduct a correlation study to check whether any correlation existed among the new dynamic coupling metrics.

a) Static CBO/Static Ca

There is generally moderate to weak correlation (-.943,-.553,.045,.149) between the two as Static Ca is based on calls received by a class whereas Static CBO is based on the calls sent out by the class. The results for the test cases System01 and System02 show a stronger negative correlation that indicates an incidental inverse proportionality between the two metrics. This strong inverse correlation indicates a relationship in which one metric’s values decrease when the other metric’s values increase and vice-versa.

b) Static CBO/Static KSC

There is a weak correlation (-.379,.465,.037,.161) between the two as SKSC is based on the calls received from other classes whereas Static CBO is based on the calls sent from the class. Their differences go further as SKSC is a call-weighed metric and Static CBO is a class-count metric. A relatively stronger inverse correlation in case of test case Institution1 is merely incidental and could be because of increasing number of received calls with decreasing number of class couples made by the class (being examined) or vice-versa.

c) Static CBO/Static KCC

There is a very high correlation (.971,.940,.891,.925) between the two metrics as both the metrics are evaluated on the basis of method calls sent out by a class. Static CBO counts the number of classes to which a class sends messages to whereas SKCC uses the number
of calls/messages sent out by a class. A strong correlation indicates that whenever the class count is increased in Static CBO, there is an increase in the number of calls in SKCC.

d) Static CBO/Static KC

There is a very high correlation (.967,.809,.799,.854) because Static CBO is strongly correlated with SKCC but is weakly correlated with SKSC i.e. a greater proportion of outgoing calls than incoming calls for a class (i.e. higher SKCC proportion than SKSC).

Also, it means that whenever SKC increases, Static CBO also increases and vice-versa.

e) Static Ca/Static KSC

There is a strong correlation (.639,.901,.836,.882) between the two as both SKSC and SCa are based on the number of method calls sent to a class. A near perfect correlation exists for all the test cases except test case 1 i.e. System01. A strong correlation indicates that an increase in number of classes accessing the methods of a class increases the number of calls sent to the class by a uniform proportion.

f) Static Ca/Static KCC

There is a moderate to weak correlation (-.995,-.463,-.137,.007) between the two as static KCC counts the calls sent whereas SCa counts the calls sent to a class. However, a near perfect negative correlation exists for System01. This correlation is incidental and could be due to an inverse proportionality between the number of classes accessing the methods of a class and number of calls sent out by a class.

g) Static Ca/Static KC

There is a moderate to high correlation (-.858,-.076,.383,.503). There exists a very strong correlation for System04, a strong correlation for System03, moderate correlation for System01 and a weak correlation for System02. The weaker correlations indicate that SKCC is more dominant than SKSC in evaluating SKC whereas, stronger correlations indicate that SKSC is more dominant than SKCC in this evaluation. This is due to the fact that SCa is strongly correlated with SKSC and weakly correlated with SKCC.

h) Dynamic Ca/Dynamic KSC

There exists a perfect correlation (1.000) between the two as both are measured at runtime and both take received method calls into account. A perfect correlation exists for all the
test cases. A strong correlation indicates that an increase in number of classes accessing the methods of a class increases the number of calls sent to the class by a uniform proportion. This dynamic correlation even improves upon the static correlation (SCa/SKSC) which also showed a very strong correlation. Especially test case1 shows a major improvement from static to dynamic correlation.

i) Dynamic Ca/Dynamic KCC

There is a weak correlation (-0.532, 0.067, 0.102, 0.260) as DCa is evaluated from the number of calls sent to a class whereas DKCC uses number of calls sent out from a class. Also DCa counts the number of classes and DKCC counts the number of calls. Comparing the dynamic correlation to the static correlation, the correlation weakens for 3 out of 4 at runtime which is again incidental as the metric counts can increase or decrease at runtime depending on the features that differentiate dynamic from static behavior (e.g. user input, polymorphism, dynamic binding etc).

j) Dynamic Ca/Dynamic KC

These two are weakly correlated (.374, 0.269, 0.251, 0.383). As compared to the static correlation (SCa/SKC), the correlation certainly goes weaker for all the test cases except System2 (but still demonstrates a weak correlation). The reason behind weaker correlations could be a more dominant role played by client classes in evaluating DKC at runtime.

k) Static KSC/Static KCC

There is a moderate to weak correlation (-0.569, -0.413, -0.114, 0.000) between the two metrics for all the four test cases. This is due to the fact that SKSC is based on calls received by a class and SKCC is based on calls out by a class. Thus these two metrics cannot have a uniform correlation.

l) Static KSC/Static KC

There is a moderate to weak correlation (-0.157, 0.026, 0.501, 0.565). There is a strong correlation between the two metrics for System01 and System02, whereas a very weak correlation for System03 and System04. The reason for a strong correlation is a more dominant role played by static server classes than client classes in evaluating SKC whereas, a weak correlation indicates dominant client classes in evaluating SKC.
m) Dynamic KSC/Dynamic KCC

There is a weak correlation (-.532,.067,.102,.260) between the two metrics for all the four test cases. This is due to the fact that DKSC is based on calls received by a class and DKCC is based on calls out by a class. Thus these two metrics cannot have a uniform correlation. The dynamic correlation results are quite similar to the static correlation (SKSC/SKCC) results except test case 4 that shows an incidental stronger correlation.

n) Dynamic KSC/Dynamic KC

There is a weak Correlation (-.374,.269,.251,.383). There is a strong correlation between the two metrics for System04, whereas a very weak correlation for System01, System02 and System03. The reason for a strong correlation is a more dominant role played by server classes than client classes in evaluating DKC at runtime, whereas a weak correlation indicates dominant client classes in evaluating DKC at runtime. It can be noted further that correlation weakens from static to dynamic analysis.

o) Static KCC/Static KC

There is a very strong correlation (.901,.899,.803,.825). The reason for a strong correlation is a more dominant role played by client classes than server classes in evaluating SKC.

p) Dynamic KCC/Dynamic KC

There is a very strong correlation (.984,.979,.989,.991). The reason for a very strong correlation is a more dominant role played by client classes than server classes in evaluating DKC. The correlation slightly strengthens if compared to the static correlation (SKCC/SKC).

5.3 Inheritance Metrics Results

After dynamic coupling metrics, focus was shifted to the dynamic inheritance metrics. Following are the new class-level dynamic inheritance metrics under study.

- Dynamic Percentage Inheritance Coupling (DPIC)
- Dynamic Method Inheritance Factor (DMIF)
- Dynamic Method Inheritance Factor – Variation 1 (DMIF1)
- Dynamic Method Inheritance Factor – Variation 2 (DMIF2)
- Dynamic Effective Number of Children (DENOC)
We have also included a static inheritance metric called MIF given by Abreu(1995) in our study to enable a static/dynamic comparison. Another equally important reason for including MIF was because 3 out of 5 dynamic inheritance metrics are modifications of MIF only. So it helps in correlation/comparative study.

5.3.1 Descriptive Analysis

A descriptive study for the test case (Educational Institution Project - System1-4) data was conducted using mean, standard deviation, percentiles for each dynamic inheritance metric. All the measures (for all the 4 systems used in the validation) exhibited sufficient variances which make them suitable candidates for further analysis. Further statistical analysis techniques required a normal data distribution. P-P plots showed a normal distribution for all the four systems. Any data that did not demonstrate a normal distribution using P-P plots was transformed computing the logarithm of each data point. Next we do a system-wise detailed analysis of test case data for each dynamic inheritance metric.

5.3.1.1 System-wise Metric Data Analysis

This subsection analyzes the dynamic metric data for each of the EIP system test case. We have divided this subsection into four parts namely System1, System2, System3 and System4. For each system, values for each of the new coupling metrics are discussed.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>DPIC(%)</th>
<th>MIF(%)</th>
<th>DMIF(%)</th>
<th>DMIF1(%)</th>
<th>DMIF2(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>System1</td>
<td>18.18</td>
<td>36.36</td>
<td>3.03</td>
<td>8.30</td>
<td>12.50</td>
</tr>
<tr>
<td>System2</td>
<td>18.36</td>
<td>33.80</td>
<td>5.63</td>
<td>16.67</td>
<td>20.00</td>
</tr>
<tr>
<td>System3</td>
<td>28.31</td>
<td>45.60</td>
<td>5.64</td>
<td>12.35</td>
<td>35.48</td>
</tr>
<tr>
<td>System4</td>
<td>29.35</td>
<td>51.00</td>
<td>2.90</td>
<td>5.85</td>
<td>26.19</td>
</tr>
</tbody>
</table>

Table 20: System values for dynamic inheritance metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Size</th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>Percentile 25%</th>
<th>Percentile 75%</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPIC</td>
<td>5</td>
<td>5.71</td>
<td>12.778</td>
<td>0.00</td>
<td>14.29</td>
</tr>
<tr>
<td>MIF</td>
<td>5</td>
<td>23.33</td>
<td>32.49</td>
<td>0.00</td>
<td>58.34</td>
</tr>
<tr>
<td>DMIF</td>
<td>5</td>
<td>1.66</td>
<td>3.71</td>
<td>0.00</td>
<td>4.15</td>
</tr>
<tr>
<td>DMIF1</td>
<td>5</td>
<td>2.50</td>
<td>5.590</td>
<td>0.00</td>
<td>6.25</td>
</tr>
<tr>
<td>DMIF2</td>
<td>5</td>
<td>10.00</td>
<td>22.361</td>
<td>0.00</td>
<td>25.00</td>
</tr>
<tr>
<td>DENOC</td>
<td>5</td>
<td>20.00</td>
<td>44.72</td>
<td>0.00</td>
<td>50.00</td>
</tr>
</tbody>
</table>

Table 21: Descriptive study of inheritance metrics for System1
Design and Validation of Dynamic Metrics for Object-Oriented Software Systems

a) System01

- DPIC has a moderate mean and standard deviation. The P-P plot (Figure 63) also shows a normal distribution. BookImpl has the highest DPIC value of 28.6% and this is the only super class in the system. Thus BookImpl is quite strongly coupled to other classes via inheritance at runtime. Higher DPIC value will possibly mean a higher difference between BookImpl’s static and dynamic afferent coupling metric values (mostly an increase in metric value from static to dynamic analysis) whereas Ca values for other classes remain same or drop at runtime. Thus amount of active inheritance is expected to increase the class-level coupling at runtime. Thus, reducing the external quality attributes like maintainability, testability and portability at runtime. Reusability is however expected to increase with active inheritance at runtime. The overall DPIC for System1 comes out to be 18.2% which means that System1 executes about 18% of inheritance based calls out of the total calls at runtime. DENOC is showing higher values for both mean and standard deviation. This is primarily because of BookImpl superclass that has a 100% value at runtime.

- MIF has highest mean value. Standard deviation is also on the higher side. These high values come from its static nature as well as the high MIF values for classes IssueInfoImpl (66.7%) and LibraryImpl (50%). System-level value of MIF is 36.3% which is quite high. DMIF has the lowest mean, deviation and percentile values among all the inheritance metrics. This is due to the fact that DMIF computes the percentage of used inherited methods out of the largest set (total methods that could be invoked from the class) among all the dynamic inheritance metrics. A low mean value also indicates that a very low percentage of inherited methods are used at runtime, hence leading to a low overall inheritance in the class objects. Class IssueInfoImpl is the only class with a non-zero DMIF (8.3%), DMIF1 (12.5%) and DMIF2 (50%) values. This is because it’s the only subclass that is accessed at runtime. Thus 12.5% of its total inherited methods are accessed at runtime, hence making a decent use of inheritance coupling at runtime. Its DMIF2 value of 50% shows that out of total methods used at runtime 50% are inherited from its superclasses. A high DMIF2 value also indicates a good balance between the usage of inherited methods and its own methods. DMIF, DMIF1 and DMIF2 have very
low system-level values for the System1 with DMIF having the lowest (3.03%). These values show a very low actual inheritance utilized by System1 at runtime.

Figure 63: P-P plots of inheritance metrics for System1
b) System02

DPIC metric shows a moderate mean but a high standard deviation. Higher standard deviation indicates towards a good distribution of values among classes. Overall data was found to be normal. DPIC for the whole system is 18.4%. EmployeeInfoImpl has the highest DPIC value of 100%. This means that EmployeeInfoImpl is only accessed via its subclasses at runtime and thus is actually totally dependent upon its subclasses to work. Thus subclasses in this case would act as an external interface for the super class. Class BookImpl has also got a high DPIC value of 24.14%. A high DPIC value will possibly mean a higher difference between BookImpl’s static and dynamic afferent coupling metric values (mostly an increase in metric value from static to dynamic analysis) whereas Ca values for other classes remain same or drop at runtime. Thus amount of active inheritance is expected to increase the class-level coupling at runtime. Thus, reducing the external quality attributes like maintainability, testability and portability at runtime. Reusability is expected to increase with active inheritance at runtime. It can also be observed that it would take a lot more effort to test a class that is using inheritance to be accessed (and in turn a method call) at runtime than a class that is accessed directly. This is due to the fact that while testing the inheritance dependent class we need to test two or more, and not one, classes whereas just one class needs to be tested for a class being accessed directly. DENOC has a high standard deviation from mean. A higher mean and standard deviations are primarily because of high DENOC values for the superclasses in the system.

- Static metrics MIF has the highest mean of 23.38 and a high standard deviation indicating a healthy distribution of values across the classes. MegazineImpl class shows the highest MIF value of 72%. DMIF again has the lowest mean, standard deviation and percentile values among all the inheritance metrics. The highest

<table>
<thead>
<tr>
<th>Metric</th>
<th>Size</th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>Percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25%</td>
</tr>
<tr>
<td>DPIC</td>
<td>10</td>
<td>12.41</td>
<td>31.696</td>
<td>.00</td>
</tr>
<tr>
<td>MIF</td>
<td>10</td>
<td>23.38</td>
<td>31.16</td>
<td>.00</td>
</tr>
<tr>
<td>DMIF</td>
<td>10</td>
<td>4.23</td>
<td>8.13</td>
<td>.00</td>
</tr>
<tr>
<td>DMIF1</td>
<td>10</td>
<td>7.50</td>
<td>15.811</td>
<td>.00</td>
</tr>
<tr>
<td>DMIF2</td>
<td>10</td>
<td>11.50</td>
<td>19.444</td>
<td>.00</td>
</tr>
<tr>
<td>DENOC</td>
<td>10</td>
<td>15.00</td>
<td>33.74</td>
<td>.00</td>
</tr>
</tbody>
</table>

Table 22: Descriptive study of inheritance metrics for System2
Figure 64: P-P plots of inheritance metrics for System2
DMIF value is registered by LibraryImpl. The class also has a very high DMIF1 value of 50%. Thus it utilizes half of the total number of inherited classes at runtime. Classes IssueInfoImpl and MegazineImpl also have healthy DMIF1 values of 12.5% each. IssueInfoImpl has the highest DMIF2 value of 50% followed by LibraryImpl (40%) and MegazineImpl (25%). A high DMIF2 value indicates a high degree of inheritance coupling at runtime and also shows a good balance between the usage of inherited methods and its own methods. System-level value of MIF is 33.8% which is quite high. DMIF, DMIF1 and DMIF2 have very low system-level values for the System1 with DMIF having the lowest (5.63%). These values show a very low actual inheritance utilized by System1 at runtime.

c) System03

- DPIC shows low mean and low 75 percentile indicating that there are very few superclasses utilizing inheritance at runtime. BookImpl has the highest DPIC value of 76.67%. Thus BookImpl is quite strongly coupled to other classes via inheritance at runtime. Other classes having a non-zero DPIC value are EmployeeInfoImpl (50%), Student (48.89%) and StudentAthlete (30%). Higher DPIC values will possibly mean higher differences between classes’ static and dynamic afferent coupling metric values (mostly an increase in metric value from static to dynamic analysis) whereas Ca values for other classes remain same or drop at runtime. The only exception is the class Student for which the Ca value drops at runtime. This is because it is mostly accessed via inheritance by the methods of class Institution2 and most of the classes that send static method calls to it doesn’t get their method calls executed at runtime. Thus amount of active inheritance is expected to increase the class-level coupling at runtime in most of the cases. Thus, reducing the external quality attributes like maintainability, testability and portability at runtime.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Size</th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>Percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25%</td>
</tr>
<tr>
<td>DPIC</td>
<td>30</td>
<td>6.85</td>
<td>18.81</td>
<td>.00</td>
</tr>
<tr>
<td>MIF</td>
<td>30</td>
<td>25.15</td>
<td>35.26</td>
<td>.00</td>
</tr>
<tr>
<td>DMIF</td>
<td>30</td>
<td>6.04</td>
<td>14.61</td>
<td>.00</td>
</tr>
<tr>
<td>DMIF1</td>
<td>30</td>
<td>5.42</td>
<td>10.73</td>
<td>.00</td>
</tr>
<tr>
<td>DMIF2</td>
<td>30</td>
<td>11.94</td>
<td>22.71</td>
<td>.00</td>
</tr>
<tr>
<td>DENOC</td>
<td>30</td>
<td>7.50</td>
<td>21.9</td>
<td>.00</td>
</tr>
</tbody>
</table>

Table 23: Descriptive study of inheritance metrics for System3
Reusability is, however, expected to increase with active inheritance at runtime. DPIC for the whole System3 is 28.31% which is higher than the previous two systems. DENOC has descriptive analysis data that is quite similar to DPIC. This is expected as both measure dynamic inheritance from a superclass perspective.

**Figure 65: P-P plots of inheritance metrics for System3**
MIF again has the highest mean and standard deviation. The highest MIF value of 83% means a high static inheritance in the class. The system MIF is 45.60% which is higher than the previous two systems. This shows an increase in percentage static inherited methods in System3 as compared to System1 and System2. StudentRepresentative class has the highest DMIF1 value of 37.5%. Classes LibraryImpl, StudentAthlete, StudentCricketer, StudentTennisPlayer also show healthy DMIF1 values of 25% each. StudentRepresentative also show the highest DMIF2 value of 75%. This shows a very high percentage of inherited methods being accessed at runtime w.r.t. the total methods accessed at runtime. DMIF and DMIF1 system values are still low for System3 but DMIF2 value is high at 35.5%. A high DMIF2 system value indicates a high percentage of inherited methods among total methods accessed at runtime for all the classes of the system.

d) System04

DPIC’s mean and standard deviation show normal distribution that is confirmed by P-P plot. System DPIC comes out to be quite substantial at 29.35%. Student class has the highest DPIC value of 55.77% whereas classes BookImpl, EmployeeInfoImpl and Club are not far behind with DPIC values of 54.67%, 50% and 50% respectively. The only other class that has a non-zero value is the class StudentAthlete (35.71%). Higher DPIC values will possibly mean higher differences between classes’ static and dynamic afferent coupling metric values (mostly an increase in metric value from static to dynamic analysis) whereas Ca values for other classes drop at runtime. The only exception is the class Student for which the Ca value drops at runtime. This is because it is mostly accessed via inheritance by the methods of class Institution2 and most of the classes that send static method calls to it doesn’t get their method calls executed at runtime. Thus

<table>
<thead>
<tr>
<th>Metric</th>
<th>Size</th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>Percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25% 75%</td>
</tr>
<tr>
<td>DPIC</td>
<td>47</td>
<td>5.24</td>
<td>15.530</td>
<td>.00 .00</td>
</tr>
<tr>
<td>MIF</td>
<td>47</td>
<td>35.67</td>
<td>35.964</td>
<td>.00 73.33</td>
</tr>
<tr>
<td>DMIF</td>
<td>47</td>
<td>1.83</td>
<td>4.49</td>
<td>.00 .00</td>
</tr>
<tr>
<td>DMIF1</td>
<td>47</td>
<td>3.99</td>
<td>9.436</td>
<td>.00 .00</td>
</tr>
<tr>
<td>DMIF2</td>
<td>47</td>
<td>8.16</td>
<td>19.073</td>
<td>.00 .00</td>
</tr>
<tr>
<td>DENOC</td>
<td>47</td>
<td>5.14</td>
<td>17.9</td>
<td>.00 .00</td>
</tr>
</tbody>
</table>

Table 24: Descriptive study of inheritance metrics for System4
amount of active inheritance is expected to increase the class-level coupling at runtime in most of the cases. Thus, reducing the external quality attributes like maintainability, testability and portability at runtime. Reusability is however expected to increase with active inheritance at runtime. DENOC has a very similar
reading to that of DPIC because both metrics measure the superclass runtime inheritance.

- **MIF metric** has the highest mean and standard deviation as compound to the previous three systems. MIF value for System4 is highest among all the systems at 51.0%. Comparing all the four systems, there is a clear increase of MIF system values with an increase in system size. This indicates that the number of inherited methods in the systems increase with the number of classes. DMIF, DMIF1 and DMIF2 have low means and standard deviations indicating an overall lesser usage of inherited methods at runtime. **StudentRepresentative** class has the highest DMIF1 value of 37.5%. Classes **LibraryImpl, StudentAthlete, StudentCricketer, StudentTennisplayer** and **EcoFriendlyClub** also show healthy DMIF1 values of 25% each. **StudentRepresentative** also show the highest DMIF2 value of 75%. This shows a very high percentage of inherited methods being accessed at runtime w.r.t. the total methods accessed at runtime. DMIF and DMIF1 values for System4 are low whereas that of DMIF2 is quite high.

There is a clear decrease of system-level dynamic inheritance metric values with size of the system as shown in Table 20. Thus where static system MIF increases with an increase in number of classes in the system, dynamic MIF metrics (DMIF, DMIF1, DMIF2) are generally found decreasing with number of classes of the system indicating a decreasing utilization of inherited methods, and in turn, overall inheritance in the system. On the other hand, DPIC(system) also increases with size (or number of classes) of the system opposed to the nature shown by other dynamic inheritance metrics. This means that as system size increases, percentage of calls sent to the superclass methods are increasing, but percentage of inherited methods executing those calls are decreasing with size i.e. lesser inherited methods handling more method calls at runtime.

### 5.3.2 Correlation Study

#### 5.3.2.1 Subclass Metrics

MIF, DMIF, DMIF1 and DMIF2 are metrics meant to measure the amount of inheritance demonstrated by the subclasses. So being subclass metrics there might be some correlations existing amongst these metrics. A Pearson correlation study was hence conducted over these metrics to capture any existing correlation.
a) MIF - DMIF/DMIF1/DMIF2

MIF shares a weak correlation with DMIF, DMIF1 and DMIF2 for test cases System2, System3 and System4 as shown in Table 25. System1 has not been considered because of insufficient data. The low correlations in case of other three systems indicates that number

<table>
<thead>
<tr>
<th>Test Case</th>
<th>MIF/DMIF</th>
<th>MIF/DMIF1</th>
<th>MIF/DMIF2</th>
<th>DMIF/DMIF1</th>
<th>DMIF/DMIF2</th>
<th>DMIF1/DMIF2</th>
<th>DPIC/DENOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>System2</td>
<td>Corr</td>
<td>-.035</td>
<td>-.163</td>
<td>.491</td>
<td>.992</td>
<td>.622</td>
<td>.553</td>
</tr>
<tr>
<td></td>
<td>Sig</td>
<td>.965</td>
<td>.837</td>
<td>.509</td>
<td>.008</td>
<td>.378</td>
<td>.447</td>
</tr>
<tr>
<td>System3</td>
<td>Corr</td>
<td>.447</td>
<td>.282</td>
<td>.386</td>
<td>.700</td>
<td>.634</td>
<td>.904</td>
</tr>
<tr>
<td></td>
<td>Sig</td>
<td>.169</td>
<td>.401</td>
<td>.241</td>
<td>.016</td>
<td>.036</td>
<td>.000</td>
</tr>
<tr>
<td>System4</td>
<td>Corr</td>
<td>.125</td>
<td>.076</td>
<td>.095</td>
<td>.856</td>
<td>.875</td>
<td>.923</td>
</tr>
<tr>
<td></td>
<td>Sig</td>
<td>.553</td>
<td>.717</td>
<td>.651</td>
<td>.000</td>
<td>.000</td>
<td>.000</td>
</tr>
</tbody>
</table>

Table 25: Correlation study of inheritance metrics

of available inherited methods before runtime and number of inherited methods actually used at runtime may not be inter-dependent. Thus an increase in the number of available inherited methods may or may not increase/decrease the actual usage of such methods at runtime.

b) DMIF/DMIF1

Pearson correlation between DMIF and DMIF1 shows high correlation values for System2, System3 and System4 as shown in Table 25. This high correlation is because both DMIF and DMIF1 are based on the percentage of number of inherited methods used at runtime. Although this percentage, for the two metrics, is taken over two different static counts (i.e. total number methods and number of inherited methods), but a high correlation indicates that the percentage of number of inherited methods among the total number of methods must be on the higher side which is proved by the MIF values. Another reason for an increase in the correlation significance with system size could be because of the increasing percentage of classes that do not utilize their inherited methods at all and hence result in zero metric values.

c) DMIF/DMIF2

There is a weak correlation for System2 whereas it is stronger for System3 and System4 as shown in Table 25. Although both the metrics are based on the number of inherited methods used at runtime, the domains, over which their respective percentages are taken,
are different (For DMIF, it is the static total number of methods and for DMIF2, it is dynamic actual number of methods used at runtime) and are not interdependent. The key reason for an increase in the correlation significance with system size could be because of the increasing percentage of classes that do not utilize their inherited methods at all and hence result in zero metric values.

d) DMIF1/DMIF2

Table 25 shows a weak correlation for System2 whereas it is quite strong for System3 and System4. Although both the metrics are based on the number of inherited methods used at runtime, the domains, over which their respective percentages are taken, are different (For DMIF1, it is the static total number of inherited methods and for DMIF2, it is dynamic actual number of methods used at runtime) and are not interdependent. Again the key reason for an increase in the correlation significance with system size could be because of the increasing percentage of classes that do not utilize their inherited methods at all.

5.3.2.2 Superclass Metrics

a) DPIC/DENOC

There is a low-moderate correlation for System3 and System4 of EIP project as shown in Table 25. This is due to the fact that DPIC is based on the number of calls to superclass via subclasses whereas DENOC is based on the number of subclasses used to send those calls at runtime. As the number of calls is not dependent upon the number of subclasses used to forward those calls, hence no strong correlation exists. Whatever correlation that exists, is because of a uniform distribution of incoming methods calls among the subclasses in the given test cases.

5.4 Correlating Coupling/Inheritance Metrics

A major objective of this research was to study the impact of inheritance on coupling at runtime. This section discusses the studies conducted to trace such an impact. This section has been divided into two major subsections. The first subsection analyzes the system-level dynamic inheritance results for each metric and tries to extract any existing effect of inheritance on coupling reflected by these results. In the second subsection we conduct a correlation study comprising of dynamic afferent coupling (DCa) and dynamic inheritance metrics to empirically validate any existing correlation between dynamic coupling and
dynamic inheritance metrics. We selected afferent coupling for the purpose because all the inheritance metrics are based upon calls received and not the calls sent as in DCBO.

5.4.1 General Analysis

a) MIF

MIF is a static metric that measures the percentage of number of inherited methods to the total number of methods of a class. Figure 67 shows MIF as percentage of inherited methods for all the four test case systems.

System1 inherits about 36% of the total methods. This leaves about 64% methods that are non-inherited or classes’ own methods. The MIF value for System2 (34%) is quite same as System1 whereas this value goes considerably up for System3 (46%) and System4.

![Figure 67: Inheritance impact - MIF](image_url)
(51%). Thus it is observed that with an increase in the number of classes in the system there is a general increase in the percentage of inherited methods in a system. Although MIF does not measure coupling still it shows the amount of methods that are capable of exhibiting interaction based coupling and those that exhibit inheritance based coupling. Thus an increase in the percentage of inherited methods in a system would lead to an increase in such methods that can demonstrate inheritance based coupling at runtime and decrease in direct interaction based methods, thus lowering the direct coupling at runtime. This would further result in wider variations from static to dynamic values at runtime as inheritance is a runtime feature of an object-oriented language.

b) DMIF

DMIF measures the percentage inherited methods that are invoked at runtime. It is the first variation of static MIF metric. DMIF for System1 to System4 are shown in the pie charts.

![Pie charts showing Inheritance Impact - DMIF for System1 to System4](image)

**Figure 68: Inheritance impact - DMIF**
in Figure 68. For System1 and System4, the actual used inherited methods at runtime are just 3% of the total methods that can be invoked from all the classes of the system. For System2 and System3, this value is slightly higher at 6%. Thus generally actual percentages of inherited methods used at runtime are quite low indicating a low inheritance-based coupling in the system. This metric improves upon the static MIF metric to indicate only those inherited methods that are actually invoked at runtime. It can be seen that corresponding MIF values are quite high that indicates that the systems are strongly capable of demonstrating inheritance-based coupling out of the total coupling at runtime, whereas DMIF extracts only the methods that are used at runtime. Thus DMIF’s findings contradict that of static MIF and shows the actual results in which the direct coupling is much more prominent than the inheritance-based indirect coupling.

e) DMIF1

DMIF1 is finds the effective inheritance at runtime by computing the percentage of used
Design and Validation of Dynamic Metrics for Object-Oriented Software Systems

inherited methods to the total inherited methods for all classes. Thus it is the percentage of DMIF taken over MIF and could be used to compare the two. As shown in pie charts in Figure 69, apart from System1 that has DMIF1 value of 8%, the value reduces with the size of the system (System2-17%, System3-12%, System4-46%). DMIF1 does not include direct coupling but analyzes the dynamic coupling’s inheritance part. Hence it works within dynamic inheritance to capture the actual inheritance executed. Again DMIF1 seems to decrease whenever static MIF increases. DMIF1 and DMIF have quite similar values as DMIF1 directly depends upon DMIF for its evaluation.

d) DMIF2

DMIF2 analysis dynamic inheritance from the perspective of number of methods used at runtime. DMIF2 percentages (as shown in Figure 70) seem to be higher than DMIF that

Figure 70: Inheritance impact – DMIF2
works with total number of methods (invoked /not invoked at runtime). Thus in DMIF, where there seems to be a very passive role of inherited methods at runtime, the actual role in much more prominent when studied in relation to dynamic coupling in DMIF2.

e) DPIC

DPIC is a call-weighed metric that finds the percentage of inheriteane-based received calls to the total number of calls received by all the classes. There is a clear increase in DPIC (System1-18%, System2-18%, System3-28%, System4-29%) with an increase in number of classes of the system. This indicates a rise in dynamic inheritance out of the total dynamic coupling at runtime. This also suggests that with increase in system size, the impact of inheritance on the overall coupling also increases. Comparing it with DMIFs,

![Figure 71: Inheritance impact – DPIC](image-url)
generally a decline in values was recorded w.r.t. system size and values for all the systems were generally much lower than that of DPIC. The reason seems to be the increasingly higher frequency of calls received by a limited number of inherited methods as system size (i.e. number of classes) increases.

5.4.2 Correlation Study

There is generally a weak correlation (Table 26) existing between DCa and the dynamic inheritance metrics because the number of classes accessing the methods of a class cannot influence the number of inherited methods used via class at runtime and vice-versa. The number of calls received by the methods of a class could be sent from any number of classes, hence weakening the relation between DCa and dynamic inheritance metrics. Higher correlation values for System4 are mainly due to higher number of inactive classes, having zero metric values at runtime. Test case System1’s execution data was found to be incapable in capturing this relationship whereas System2 too fails to do so for DPIC/DCa and DENOC/DCa because of the insufficient runtime data.

<table>
<thead>
<tr>
<th>Test Cases</th>
<th>DPIC/DCa</th>
<th>DMIF/DCa</th>
<th>DMIF1/DCa</th>
<th>DMIF2/DCa</th>
<th>DENOC/DCa</th>
</tr>
</thead>
<tbody>
<tr>
<td>System2</td>
<td>Corr.</td>
<td>.920</td>
<td>.962</td>
<td>.345</td>
<td>.655</td>
</tr>
<tr>
<td></td>
<td>Sig.</td>
<td>.080</td>
<td>.038</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System3</td>
<td>Corr.</td>
<td>.244</td>
<td>.439</td>
<td>.480</td>
<td>.529</td>
</tr>
<tr>
<td></td>
<td>Sig.</td>
<td>.692</td>
<td>.769</td>
<td>.177</td>
<td>.135</td>
</tr>
<tr>
<td>System4</td>
<td>Corr.</td>
<td>-.204</td>
<td>.646</td>
<td>.694</td>
<td>.248</td>
</tr>
<tr>
<td></td>
<td>Sig.</td>
<td>.661</td>
<td>.000</td>
<td>.000</td>
<td>.592</td>
</tr>
</tbody>
</table>

**Table 26: Correlation study of inheritance metrics and Ca metric**

5.5 Additional Metrics

5.5.1 Constructor-based Automatic Initialization Metrics

The new coupling metrics do not consider calls to constructors while evaluation. So there must be some measure that takes usage of constructors of a class into account to estimate the amount of automatic initialization inhibited by the objects of a class at runtime. We put such metrics into a different category called Automatic Initialization metrics. Such metrics will measure the amount of automatic initialization of a class at runtime. We have proposed one such metric in this work. It is called Dynamic Percentage Constructor Calls Received (DPCCR).
5.5.1.1 Descriptive Statistics

Descriptive data shown in Table 27 for DPCCR showed normal means and standard deviations for all the systems. Top 25% of the classes also had high values especially for System1, 75 percentile came out to be 75% which is very high. Percentile values for other systems are also quite high. This means that the top 25% of classes in each of the systems,

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>Percentile 25%</th>
<th>Percentile 75%</th>
</tr>
</thead>
<tbody>
<tr>
<td>System 1</td>
<td>41.00</td>
<td>37.483</td>
<td>12.50</td>
<td>75.00</td>
</tr>
<tr>
<td>System 2</td>
<td>29.20</td>
<td>20.436</td>
<td>12.50</td>
<td>50.00</td>
</tr>
<tr>
<td>System 3</td>
<td>21.77</td>
<td>23.666</td>
<td>.00</td>
<td>50.00</td>
</tr>
<tr>
<td>System 4</td>
<td>13.00</td>
<td>20.233</td>
<td>.00</td>
<td>30.00</td>
</tr>
</tbody>
</table>

Table 27: Descriptive analysis - DPCCR

Figure 72: P-P plots of DPCCR for all Systems
have a lot of object initializations at runtime but very less use of those objects i.e. very few methods calls received by the objects of the classes. This could be because of bad design or deliberate high automatic initialization preferred at runtime. It can also be observed that the mean values decrease with increase in system size i.e. number of classes. This indicates that as the number of classes increases the amount of class methods (other than constructors) also increase at runtime and hence the classes become lesser automated. The data that was not found to be normal using P-P plots was normalized by using logarithm of the data. The normal P-P plots are shown in Figure 72.

5.5.2 Method-level Coupling Metrics

5.5.2.1 Descriptive analysis

Descriptive study worked on mean and standard deviation data for all the four systems. P-P plots showed normal distribution any data that did not show the normal distribution was normalized by using logarithm of data, as shown in Figures 73-74. Low mean and standard deviation values for MQFS indicate a very few number of invocations for each method or improper/skewed distribution of method calls among the methods at runtime. It can be noticed that the mean values for MAM metric is generally two times the MQFS means. This is because MQFS is based on the methods to which a particular method sends calls, whereas MAM is based on sum of calls sent out and calls received by the method. Thus as MAM is a call-weighted metric and takes into account incoming as well as outgoing calls, its values are bound to be higher than MQFS. Overall all the systems show low mean values for both the metrics indicating low method usage at runtime.

<table>
<thead>
<tr>
<th></th>
<th>Metric</th>
<th>Size</th>
<th>Mean</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>System1</td>
<td>MQFS</td>
<td>21</td>
<td>.52</td>
<td>2.182</td>
</tr>
<tr>
<td></td>
<td>MAM</td>
<td>21</td>
<td>1.10</td>
<td>2.406</td>
</tr>
<tr>
<td>System2</td>
<td>MQFS</td>
<td>48</td>
<td>1.02</td>
<td>3.118</td>
</tr>
<tr>
<td></td>
<td>MAM</td>
<td>48</td>
<td>2.06</td>
<td>4.319</td>
</tr>
<tr>
<td>System3</td>
<td>MQFS</td>
<td>108</td>
<td>1.11</td>
<td>4.378</td>
</tr>
<tr>
<td></td>
<td>MAM</td>
<td>108</td>
<td>2.25</td>
<td>7.463</td>
</tr>
<tr>
<td>System4</td>
<td>MQFS</td>
<td>177</td>
<td>.85</td>
<td>3.899</td>
</tr>
<tr>
<td></td>
<td>MAM</td>
<td>177</td>
<td>1.71</td>
<td>6.311</td>
</tr>
</tbody>
</table>

Table 28: Descriptive analysis – MQFS, MAM
5.6 Result Summary

5.6.1 Coupling Metrics

- Four new dynamic coupling metrics have been proposed
  - Dynamic Afferent Coupling (DCa)
  - Key Server Class (KSC)
  - Key Client Class (KCC)
  - Key Class (KC)
- Static and Dynamic CBO metrics are strongly correlated for 3 out of 4 test cases of the sample application.
Static and Dynamic Ca metrics are strongly correlated for 3 out of 4 test cases of the sample application.

Static CBO is very strongly correlated to static KCC for all the four test cases whereas dynamic CBO is very strongly correlated to dynamic KCC for 3 out of 4 test cases and is strong for the remaining one test case.

Dynamic KCC is very strongly correlated to dynamic KC for all the four test cases whereas static KCC is very strongly correlated to static KC for 3 out of 4 test cases and is strong for the remaining one test case.

Static KSC has moderate to high correlation with static KC, the correlation weakens to a moderate-level at runtime.

The previous two conclusions indicate a major role played by KCC in evaluating KC.

---

**Figure 74: P-P plots of MAM for all Systems**

- Static and Dynamic Ca metrics are strongly correlated for 3 out of 4 test cases of the sample application.
- Static CBO is very strongly correlated to static KCC for all the four test cases whereas dynamic CBO is very strongly correlated to dynamic KCC for 3 out of 4 test cases and is strong for the remaining one test case.
- Dynamic KCC is very strongly correlated to dynamic KC for all the four test cases whereas static KCC is very strongly correlated to static KC for 3 out of 4 test cases and is strong for the remaining one test case.
- Static KSC has moderate to high correlation with static KC, the correlation weakens to a moderate-level at runtime.
- The previous two conclusions indicate a major role played by KCC in evaluating KC.
• Although the highest system-level DCBO of 71.17% was recorded for System4, DCBO for other systems are not far behind. Thus there is no huge variation in percentage of dynamic coupling at runtime and in turn the dynamic cohesion.

• PAC decreases, whereas the number of inactive classes increases, with increase in number of classes, where System2 has the highest percentage of 100%.

5.6.2 Inheritance Metrics

• Five new dynamic inheritance metrics are proposed:
  o Dynamic Percentage Inheritance Coupling (DPIC)
  o Dynamic Method Inheritance Factor (DMIF)
  o Dynamic Method Inheritance Factor – Variation 1 (DMIF1)
  o Dynamic Method Inheritance Factor – Variation 2 (DMIF2)
  o Dynamic Effective Number of Children (DENOC)

• MIF shares a weak correlation with each of the DMIF, DMIF1 and DMIF2 for all the systems. This indicates wide possible differences between the static and dynamic metrics.

• DMIF correlation with DMIF1 and DMIF2 gets stronger with an increase in number of classes.

• DMIF1 and DMIF2 are highly correlated for System3 and System4 whereas a weak correlation exists for system2.

• DPIC and DENOC are moderately correlated for System3 and System4.

• System MIF increases whereas system DMIFs decrease, with an increase in number of classes. Thus static and dynamic inheritance measures seem to behave quite differently.

• System DPIC increases with number of classes. This could be a result of increasingly higher frequency of calls received by a limited number of inherited methods as number of classes increases.

5.6.3 Coupling-Inheritance Relation

• For every class having DMIF1 or DMIF2>0 and DPIC=0, there is either no change or a decrease in Ca metric from static to dynamic analysis. Thus for a class (that is not a superclass) that uses its inherited methods at runtime, actual afferent coupling remains same or decreases considerably.
• For every class having DPIC>0 and DMIF1=DMIF2=0, there is either no change or an increase in Ca metric from static to dynamic analysis. Thus if a class receives calls via its subclasses at runtime, its incoming coupling (Ca) is found to remain same or rise from static to dynamic analysis.

• Percentage of number of classes using inheritance at runtime (either as superclass or subclass) is directly correlated to the correlation between DCa/DCBO. It indicates that with increase of amount of inheritance at runtime, the correlation between DCa and DCBO strengthens. In other words, a strengthening correlation between DCa and DCBO indicates an increase in the number of classes using inheritance at runtime. Thus inheritance proves to be one of the reasons behind a strong DCa/DCBO correlation.

• If ENOC is increasing with Ca then that would indicate a more number of classes accessing the class via its subclasses than directly calling the methods of the class.

• There is generally a weak correlation existing between DCa and the dynamic inheritance metrics because the number of classes accessing the methods of a class cannot influence the number of inherited methods used via class at runtime and vice versa. However a rise in DCa with an increase in any of the DMIF metrics would mean that every class accessing the methods of a particular class not only accesses its inherited methods but also that each class accesses a new inherited method(/s) not accessed by the other classes.

**Chapter Summary**

This chapter covers the statistical validation of the proposed dynamic metric suite. The chapter is divided into two major sections: dynamic coupling metric results and dynamic inheritance metric results. Statistical analysis is divided into two parts: descriptive analysis and correlation study. Descriptive analysis is discussed system or test-case wise. Both the class-level and system-level metric values were analyzed for all the metrics. Normality of data was checked using P-P normality plots. Correlation study was conducted using *Pearson’s Correlation* or *Product Moment Correlation* for studying any correlation existing between any of the proposed dynamic coupling metrics and the dynamic CBO coupling metric. Similarly a correlation study was also conducted among all the proposed dynamic coupling metrics and all the proposed dynamic inheritance metrics. Additional metrics were also statistically analyzed. The most important analysis in this chapter is the
correlation study among the dynamic afferent coupling (DCa) metric and various dynamic inheritance metrics. This was done to find out any existing relationship between coupling and inheritance at runtime, and also to figure out any impact of inheritance on coupling at runtime. The chapter concludes with a summary of the metric analysis based conclusions. These conclusions are also divided into three categories: coupling, inheritance and coupling-inheritance.