CHAPTER 6

COUPLING AND COHESION METRICS IN JAVA FOR
ADAPTIVE REUSABILITY RISK REDUCTION

6.1 INTRODUCTION

The SDLC has five phases, namely ‘Requirement Analysis’, ‘Design’, ‘Implementation’, ‘Testing’ and ‘Maintenance’. The First phase ‘Requirement Analysis’ is used for preparing the SRS (Software Requirement Specification) document through elicitation process with customer. This document is an input to the ‘Design Phase’ that helps the design team to
create the architecture of the software. By using the software architecture, the
development team can then generate the source code through the
‘Implementation’ phase. The source code will then be tested to ensure that the
requirements of the customers that are specified in SRS document are met.
After confirmation of the requirements and functionalities of the software, the
software will then be deployed to the customer side to commence their
activities using the product. While the customers process their product, they
may notice some irrelevances to what they expect. The irrelevance may be
due to some undetected bugs and those bugs should be rectified. The SDLC
process does not end with the deployment of the product but continues
through the Maintenance phase.

‘Reusable Estimation Effort’ techniques are used to estimate the
required effort for each phase of SDLC to complete its tasks as shown in
Figure 6.1.
Figure 6.1 Reusability Estimation Efforts

For example, Requirement Analysis and Design phase needs only 20% each, from the total efforts. Unlike Requirement analysis and Designing, Implementation phase has more importance as it is subjected to actions which
require sound knowledge in technical background. It takes 40% of the total efforts (Suresh Babu and Srivatsa 2009). This effort can further be reduced through the reusable components by applying the ‘ARRA’ model used in the software coding.

Consider a project containing 10 modules. In that, if five of the modules are already available in existing software products, the reusability would yield to:

i) \[ R = \frac{P}{T} \]
\[ R = \frac{5}{10} = 1/2 \]
\[ = 0.5 \text{ or } 50 \% \]

ii) \[ RE = TE - R \]
\[ = 1 - 0.5 \]
\[ = 0.5 \text{ (or) } 50 \% \]
where \( P = \) Number of reusable packages in a module, \( T = \) Total Number of identified packages in a module, \( TE = \) Total reusability estimated effort of a project, \( R = \) Required effort from total reusability estimated effort, \( RE = \) Remaining Efforts.

Hence when reusability of packages is implemented in the development of new products, 50\% efforts are reduced in the product development.

6.2 DESIGN METRICS FOR THE RESUABILITY OF PACKAGE

Design features for reuse in a software package are: Dependence Relationship, Fan-In and Fan-Out, Package Integration. They are elaborated below:

6.2.1 Package
Groups of all classes work together among themselves to accomplish a set of unified responsibilities; they are often referred to as packages. Moreover, modules and packages resemble. Each module is a well defined package in a project that is potentially useful to other projects to create a new environment. Independent packages are easier to maintain because design and implemented modification risks are limited, range of error is reduced and reusable packages are possibly used.

6.2.2 Package Dependence Relationship

The package dependence relationship comprises two components namely dependence relationship and package integration. They are explained below.
6.2.2.1 Dependence Relationship

Software Modules are integrated to provide the functionality to the whole system. The modules may be of independent or dependent form.

- Dependent modules - linked with independent module.

- Two types of ‘Dependency Relationship’: Uni-Direction and Bi-Direction are expressed pictorially.
Bi-directional

Figure 6.2 Dependence Relationship

Dependence relationship describes that any modularized system package will be communicated among them to fulfill their customer requirements with any risk factors. The comparison of dependence relationship is given in Table 6.1.
Table 6.1  Comparison of Uni-direction and Bi-direction Dependence Relationship

<table>
<thead>
<tr>
<th>Uni –Direction Dependence Relationship</th>
<th>Bi- Direction Dependence Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>● One Package is highly dependent on another package.</td>
<td>● Two packages are depending upon each other.</td>
</tr>
<tr>
<td>● Depended package alone cannot be reused</td>
<td>● One of the packages alone cannot be reusable, if any functionality matches with a new requirement, both the packages must be selected for reuse.</td>
</tr>
<tr>
<td>● Independent package reduces the wastage of memory.</td>
<td>● Memory is wasted for loading two packages in the current system development.</td>
</tr>
</tbody>
</table>

6.2.2.2  Package Integration
Interfaces denote the process of interaction among the different components in a system. The combination of data and control flow is transferred as the message from one package to another in the link. The message is then subjected to various validations and verifications.

Initially the ‘Data and Control flow’ is validated and verified followed by the Fan-In, Fan-Out calculations. Fan-In denotes the incoming requests to a package, whereas Fan-Out provides services to the other packages.

Secondly, the coupling and cohesion should be validated. Coupling describes the strength of the attachment and cohesion describes the relationship among the components. Cohesion is also used to determine the dependence among classes and procedures of a package.
6.3 ADAPTABLE ANALYSIS FOR REUSABILITY IN PACKAGE

The concept of reusability involves identifying similar patterns and reintroducing their software in a new domain to minimize the required efforts and adaptable resources for building the new software system (Anthony Finkelstein et al 1996). Reusable package resources of cohesion, coupling, stability and complexity are shown in Figure 6.3.
Figure 6.3 Metrics of Reusability

The reusable package resource metrics or characteristics are analyzed. The different metrics of Object-Oriented software components (Kuljit Kaur and Handeep Singh 2008) improve the ‘Coupling’ and ‘Cohesion’ metrics. The metrics ‘Stability’ and ‘Complexity’ of a software package are used to improve the ‘Observability’ of the software which is to be measured by giving suitable reusable packages in a current development.

6.3.1 Cohesion and Coupling Metrics for Reusability

The logical relationship among the elements or how they ‘belong to each other’ in a design unit is called cohesion. The cohesion metrics are normalized to minimum (0) or maximum (1) cohesion. When the elements are
entirely independent, they will have a minimum cohesion whereas, if each element is totally dependent on every other element, it has maximum cohesion.

Coupling denotes how closely a package is related to another, and this depends on interface. Normally, Coupling is a client-server supplier relationship among design elements and hence Import coupling (using) and Export coupling (used) can be distinguished.

Import coupling denoted by $I_v$, represents the value of the extent of a package’s usage and also its dependency on other design elements. It keeps a count on the number of services taken by a dependent module by another module.

Export coupling denoted by $E_v$, are elements which may change in forthcoming years that they may deliver a huge impact on the system, if the interface is affected by the changes.
These determination and analysis of different metrics will lead to the elimination of future risks.

**Measuring Coupling and Cohesion Risk Metrics:** Coupling and cohesion metrics are required to determine the reusability metrics. The reusability metrics have to be calculated to ensure the reusability of a package.

Coupling Value for any particular package (Cv) can be obtained by using the following Equation (6.1):

\[
C_v = E_v - I_v
\]  \hspace{1cm} (6.1)

where \(E_v\) - Export Coupling, \(I_v\) - Import Coupling
From Equation (6.1), the strength of the coupling which can be measured is presented in Table 6.2.

**Table 6.2 Measurement of Strength of Coupling between Packages**

<table>
<thead>
<tr>
<th>S.No</th>
<th>Coupling Value for the Package (Cv)</th>
<th>Export Coupling (Ev)</th>
<th>Import Coupling (Iv)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>0 (Loosely coupled)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>1 (Tightly coupled)</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The export and import couplings determine the strength of the coupling between the packages. If the export coupling is equal to that of import coupling, it shows that there are loosely coupled packages. It is tight when there is no import coupling between packages.
6.3.2 Measuring Stability Metric

An unstable package cannot be reused in future; it may not produce perfect response whenever the requirements are changed. The change of a module is dependent on the requirement of the customer. The reusability of a package is based on the stability of that package. Stability(I) can be found by using ‘Import’ and ‘Export’ type of couplings of a package. The range of stability value is [0, 1]. If the value is 0, then the package is Unstable; else it is Stable. This can be derived from Equation (6.2).

\[
\text{Stability I} = \frac{E_v}{(I_v+E_v)}
\]  (6.2)
6.3.3 Complexity of a Package

Complexity of the package is the metric which expresses the relationships or dependencies of the elements in a design unit. For example, calculating the number of method invocations inside a class determines the complexity of the class.

When the complexity of interactions is considerably high, the understandability of the specific design becomes complicated, which in turn leads to proneness of numerous faults. Testing such complex designs also becomes difficult. Hence the complexity of a design indicates very well, the quality and the management risks involved in the package.

Even in a reusable package, there are unused elements that occupy and waste the available memory space. The service and collaborations of each element in a package are taken under consideration to point out the unused
elements. As stated before, Cohesion is also thoroughly analyzed to find out the unused components in the package.

The analysis and determination of these four metrics of reusability encourages the tool design of the ARRA model.

6.4 TOOL DESIGN

In Figure 6.4, the input software is compared with the assay of software package for user requirement. It is used to identify the packages and sub-packages. After matching, the coupling and cohesion, instability and reusability are measured for the reusability of the software packages. If the match fails, with the help of enervative and destructive risks, the problem may be identified and removed from the software package.
Get the input Software Program required

Assay of Identifying Software Package

Identifying Software Element in every Package

Counting each Element and relationship of adaptability

Not Matching

Matching

Check Enervative risk and Destructive risk

Measuring Coupling, Cohesion, Instability for reusability of Software Package.

Display the risk level value of metrics of each Package in the system.

Reusable package
6.5 RESULTS AND DISCUSSION

The output of the proposed system may give the levels of coupling and cohesion of a package and the types of dependence between two packages. Package Identification attributes are package name, number of elements in a package etc. The steps involved in the process for determining the reusability is narrated below.

**Sequences and steps for the solution:**

Step 1: Determine the main package for the application system (Table 6.2).

Step 2: Determine sub package for corresponding main package (Table 6.2).
Step 3: Quantify the coupling class (Table 6.3).

Step 4: Classify import and export couplings (Table 6.4).
- To arrive at the results of step 3 and step 4, the concertinaed abstract classes are needed. Hence they are determined initially (Table 6.3).

Step 5: The metrics are segregated and tabled (Table 6.5)

Step 6: Determine the final stability (using the factors of step 5) and Reusability (Table 6.6)

Step 7: The analysis on the bench marks values similar to those explained in Table 5.7 is repeated for JAVA based project (see section 4.9.2)

From an e-pay system (example), the identified packages are presented with the result in Table 6.3.
The epayment package is dependent on five other packages such as the epayment.framework, epayment.adapter, epayment.process, epayment.request and epayment.response packages. The elements of epayment.adapter package is dependent on the results of the above mentioned package elements. For example, if the stability value of epayment.adapters is 0 then it is Unstable and it cannot be reused in future. If epayment.response has stability value of 1, then it is stable and it can be reused. This proposed tool can take a software product code as input to analyze its user defined packages, identified through modularity technique. The tool will list out all the packages and their information. For example, the table below gives the packages which are created in the ‘electronic payment’ product (in JAVA).
Table 6.3 presents the parent package and its sub packages. ‘epayment’ package is the parent and the ‘main’ package of the ‘Electronic Payment System’ has 5 sub packages.

Table 6.3 Main package and Sub package of ‘Epayment’

<table>
<thead>
<tr>
<th>ELECTRONIC PAYMENT SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>epayment</td>
</tr>
<tr>
<td>epayment.framework</td>
</tr>
<tr>
<td>epayment.adapter</td>
</tr>
<tr>
<td>epayment.process</td>
</tr>
<tr>
<td>epayment.request</td>
</tr>
<tr>
<td>epayment.response</td>
</tr>
</tbody>
</table>

Table 6.4 shows two types of coupling metric values of each
package in the given system. ‘Export’ coupling metric represents the ‘server’
class of the corresponding package. This class provides its services to other
classes in other packages. ‘Import’ coupling metric represents a client class
and it obtains the services from ‘Export’ coupling classes of other packages.

Table 6.4  ‘Coupling’ metrics of the packages in ‘Electronic Payment
System’

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Export Coupling Class</th>
<th>Import Coupling Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>epayment.framework</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>epayment.adapter</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>epayment.process</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>epayment.request</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>epayment.response</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 6.5 shows the concrete and abstract classes which are in each sub-package of the ‘Electronic Payment System’. The ‘Concrete Class’ is the main class of the corresponding package system. The ‘Abstract Class’ is the base class in which the methods are shown without definition part. It has only the method declaration. This type of classes is being defined as a source for the design of the remaining classes within the same package.

**Table 6.5 Concrete and Abstract Classes in Sub-Packages**

<table>
<thead>
<tr>
<th>S.NO.</th>
<th>Name of the sub Package</th>
<th>Concrete Class</th>
<th>Abstract Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>epayment.framework</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2.</td>
<td>epayment.adapter</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3.</td>
<td>epayment.process</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4.</td>
<td>epayment.request</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5.</td>
<td>epayment.response</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
From Table 6.6 the result shows each package of epayment system along with its corresponding position values of ‘Import Coupling’, ‘Export Coupling’ and ‘Stability’ metrics. These results will be considered for judging whether those packages will be reusable or non reusable in further incoming projects of a development team.

**Table 6.6 Metrics Table**

<table>
<thead>
<tr>
<th>S.No</th>
<th>Package Name</th>
<th>Coupling</th>
<th>Stability Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Ev</td>
<td>Iv</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>epayment.framework</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>epayment.adapters</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>epayment.request</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>epayment.response</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>epayment.processor</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 6.7 shows the ‘Stability’ value of individual every package in ‘Electronic Payment’ System and whether its value is ‘0’ or ‘1’. So it will be reusable, when RE=1 and there will not be any risk for the development team and if RE =0, it will lead to some risks for the development team.

Table 6.7 Stability, Reusability and Risk of each Package

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Stability (I)</th>
<th>Reusability (RE)</th>
<th>Risk (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>epayment.framework</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>epayment.adapter</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>epayment.request</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>epayment.response</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>epayment.process</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The dependence of a package reusability lies on the interaction among the packages in a system. High Level design patterns clearly show the concept
of reusability. Reusability packages reduce the consumption of system memory, human effort in the product development and can be used to enhance product’s quality. Reusability of packages also eliminates the difficulties in various areas and enables the product to be delivered to the customer within the time schedule.

Metrics like Cohesion, Coupling and Stability can also be used to reduce the complexity of the package. The service of a package could be studied easily using its calling methods. This promotes the confidence of the professionals to develop the product within a short span time and will permit the product’s future development and maintenance. The ARRA model will also be useful to test the communication between packages which are implemented in different programming languages such as .Net, C and C++.