CHAPTER 6
MAPPING QUALITY FACTORS WITH METRICS AND QUALITY ASSURANCE FRAMEWORK
MAPPING QUALITY FACTORS WITH METRICS AND QUALITY ASSURANCE FRAMEWORK

6.1 Mapping Quality Factors With Software Metrics

For proposing a quality assurance framework mapping of identified factors and the corresponding sub-factors with the software metrics is needed. Statistical analysis has been performed on the values of various software metrics to establish a relationship amongst the metrics. Mapping of factors with metrics provide a mechanism to evaluate software quality.

6.1.1 Quality Factors and Sub-Factors

The quality factors and sub-factors considered were based on McCall quality model, Boehm quality model, ISO 9126 quality model, Dromey quality model and Gillies relational quality model. The vital and common quality factors which were selected for proposing the quality assurance model and were extracted from the above mentioned quality models are efficiency, maintainability, portability, reliability and usability. It is appropriate to consider these internal quality factors because these are important from the developer’s
perspective. Figure 6.1 provides a brief overview of the quality factors and sub-factors which are used in proposing quality assurance model.

In total 58 quality sub-factors were identified for the study however, only 34 quality sub-factors were considered in the quality assurance model as the remaining sub-factors do not have relation with any of the selected software metrics or are not suited to the developer's perspective.
6.1.2. Quality Metrics

The software quality metrics used for proposing quality assurance model are the collection of the most popular metrics discussed in the literature survey (Chapter 2). The various metrics are taken from the well known metrics suite namely CK and MOOD suite. From among a total of ten metrics namely, WMC, RFC, CBO, NOC, DIT, AIF, MIF, PF, MHF and AHF; eight identified metrics (Chapter 4) were used in the development of the framework.

6.1.3 Mapping Quality Sub-Factors With Metrics

The relationship of quality factor with its sub-factors and further sub-factors with metrics were identified. The relationship between the various quality sub-factors and metrics is based upon the empirical validation of the metrics using software tools i.e. CKJM and EM, applied on three versions each of Apache ivy, JfreeChart and Heritrix components (Chapter 5). The statistical correlation analysis between various identified metrics provided data for the validation of metrics and for mapping of the various quality factors, sub-factors to the identified metrics.

The relationship between the quality metrics and sub-factors can be explained on the following bases:

no relation means that there is no significant relation between the metrics and quality criteria. Such metrics have not been further used for mapping.

related means that there is a significant relation between the metrics value and the quality criterion.
1. Efficiency Quality Factor

On the basis of the definition of efficiency sub-factors given in Table 4.2 and correlation values tabulated in Table 5.14 mapping was performed. The various efficiency sub-factors are time behaviour, resource behaviour, reply time, processing speed, execution efficiency and robust. The sub-factors of efficiency and related metrics are tabulated in Table 6.1.

<table>
<thead>
<tr>
<th>TABLE 6.1: SUB-FACTORS OF EFFICIENCY AND RELATED METRICS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Factor</strong></td>
</tr>
<tr>
<td>Effciency</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Interpretation of the relationship between sub-factors and metrics is drawn as follows:

a) Time Behaviour vs Related Metrics

The definition of time behaviour can be interpreted as response time, throughput and capacity to perform. Consuming time during performing task means it is related to WMC, RFC, CBO, AIF, MHF and PF. Increase in source code mean: more number of classes, more classes in depth, more responses, or more inheritance levels. More source code processing requires more time to process. The metrics WMC, RFC, CBO and AIF are positively and highly related among each other meaning thereby that when the values of these metrics increase, it worsens the time behaviour. Higher value of MHF implies lesser availability of methods and moreover, MHF and PF are highly
negatively related to other metrics thus, if MHF and PF increases time behaviour improves.

i) Time Behaviour with WMC: WMC is the sum of the complexities of the methods of a class. Classes with larger NOM will have increased number of instructions. Thus, increase in WMC will worsen the time behaviour.

ii) Time Behaviour with RFC: RFC represents set of methods that can be executed in response to a message received by an object of that class. Time behaviour will worsen with higher RFC because more communication among classes will require more and higher execution complexity.

iii) Time Behaviour with DIT: DIT may not affect time behaviour.

iv) Time Behaviour with CBO: Increase in CBO will worse time behaviour. Excessive coupling between the object classes implies that the execution is not localised but might involve many other parts of the system, thus execution during operation does not remain local, but might involve many other parts of the system. Time behaviour will become worse with increasing CBO.

v) Time Behaviour with NOC: NOC may not have impact on time behaviour.

vi) Time behaviour with AIF: Due to the fact that higher inheritance involves indirect calls thus, involves more complexity as it is likely to inherit more variables. Time behaviour has inverse relation with AIF.
**vii) Time behaviour with MHF:** Time behaviour improves with increasing MHF. Private methods restrict access resulting in faster processing.

**viii) Time behaviour with PF:** Over riding of methods reduces processing of source code, thus increased PF improves time behaviour.

**b) Resource Behaviour vs Related Metrics**

Use of resources is associated with resource behaviour. Memory utilisation, disk utilisation, input device utilisation and output device utilisation form part of resource behaviour. The capability to consume amounts and types of resources during task performing is related to WMC, RFC, CBO, AIF and MHF. Increase in use of resources is affected by more classes demanding for resources. Further, more code to process demands more memory; increase in WMC denotes more demand for resources. With an increase in responses for a class memory requirements enhance; increasing value of RFC indicates more memory needs. Coupling refers to a bond between objects, existence of a stronger bond symbolizes more demand for resources; increasing value of CBO implies more needs for resources. Availability of attributes deep in the inheritance levels requires more resources to process; AIF increase worsens resource behaviour. The metrics WMC, RFC, CBO and AIF are strongly positively related among each other. Resource behaviour worsens with increase in WMC, RFC, CBO and AIF. However, resource behaviour improves with increase in MHF due to its negative correlation value.
i) **Resource Behaviour with WMC**: Larger number of methods per class means increased size. Size indicates higher memory utilisation. Resource Behaviour will get poorer with the increase in WMC.

ii) **Resource Behaviour with RFC**: RFC indicate a set of methods that can be executed in response to a message. Higher value of RFC means more responses and will result in more memory utilisation. Resource Behaviour will get shoddier with increase in RFC.

iii) **Resource Behaviour with DIT**: DIT may not have predominant effect on resource behaviour.

iv) **Resource Behaviour with CBO**: Coupling involves other parts, devices and classes. Thus execution during operation does involve huge parts of the system. Resource Behaviour will get bad with increasing CBO.

v) **Resource Behaviour with NOC**: NOC may not have impact on resource behaviour.

vi) **Resource Behaviour with AIF**: Higher Inheritance implies more reference to memory and other resources denoting that with increase in AIF the resource behaviour deteriorates. More resources are used in case of higher inheritance.

vii) **Resource Behaviour with MHF**: Increase in method hiding factor improves resource Behaviour. Increased encapsulation reduces the load of resources being utilised. Burden on resource Behaviour will get reduced with increase in MHF.

viii) **Resource Behaviour with PF**: Resource behaviour may not be affected by PF.
c) Reply Time vs Related Metrics

Reply time is the elapsed time between the end of an inquiry or demand on a computer system and the beginning of a response. It is usually affected by WMC, RFC, CBO, AIF, MHF and PF. More source code to process requires more time; similarly increased degree of responses for a class also requires more time to process. Coupling involves bonding of objects and hence needs more time to respond. Attributes inherited imply more time to respond. On the other hand, hiding factors and overriding reduce complexities and improve reply time.

i) Reply time with WMC: Additional methods per class surely require extra processing. Reply time worsens with increasing WMC value.

ii) Reply time with RFC: With the increase in RFC, reply time also increases. Added number of responses for a class increases the time required to reply, hence deteriorates reply time.

iii) Reply time with DIT: Variation in values of DIT may not affect reply time.

iv) Reply time with CBO: Strong coupling between the objects requires more processing needs and slows down. CBO worsens reply time.

v) Reply time with NOC: NOC may not cause reply time to change.

vi) Reply time with AIF: Retrieval of attributes from depth of classes needs more efforts. Higher AIF deteriorates reply time.

vii) Reply time with MHF: Abstracted data improves reply time. As limited access is available for execution, the MHF improves reply time.
viii) Reply time with PF: Overriding improves reply time. CPU requires less time to execute the overridden code.

**d) Processing Speed vs Related Metrics**

Processing speed depends upon the time taken to complete a prescribed procedure or it relies on the actual amount of time spent working on software product to generate the result. Processing is adversely affected by; increase in source code, increase in responses, coupling and inheritance. In contrast data abstraction and polymorphism improves processing speed.

i) Processing speed with WMC: It will decline with increasing WMC as more classes require more time to process.

ii) Processing speed with RFC: It will decline with increasing RFC. Processing speed deteriorates with increasing response set size.

iii) Processing speed with DIT: With increase in DIT the processing speed decreases due to the fact that the control has to traverse deep down in the inheritance tree.

iv) Processing speed with CBO: Stronger coupling demands more processing. Increased CBO reduces processing speed.

v) Processing speed with NOC: Change in the values of NOC may not affect processing speed.

vi) Processing speed with AIF: Processing speed decreases with an increase in Inheritance levels. Processing speed will decline with increase in AIF.
vii) *Processing speed with MHF:* Processing speed will increase with higher degree of encapsulation. Rise in MHF increases processing speed.

viii) *Processing speed with PF:* Overriding reduces the complexity burden on the processing hence improving the processing speed.

e) **Execution Efficiency vs Related Metrics**

Execution efficiency is the run time performance of the component. To improve the execution efficiency the developer during the development stage would attempt to restrict the complexity.

i) *Execution Efficiency with WMC:* Higher values of WMC require more execution efforts. Execution efficiency will decline with increase in WMC.

ii) *Execution Efficiency with RFC:* Execution efficiency shall decline with increase in RFC.

iii) *Execution Efficiency with DIT:* Higher values of DIT reduce execution efficiency because classes existing deep down in the inheritance tree are accessed slowly and hence reduce the efficiency.

iv) *Execution Efficiency with CBO:* CBO worsens the execution power. Highly coupled objects need more time to perform.

v) *Execution Efficiency with NOC:* Execution efficiency may not be affected by NOC.

vi) *Execution Efficiency with AIF:* Deeper access of attributes and methods require more hard work. Execution efficiency shall decline with increase in AIF.
vii) Execution Efficiency with MHP: Increasing values of MHP means more encapsulation and less access to attributes and methods. Efficiency would increase with more encapsulation.

viii) Execution Efficiency with PF: The power of efficiency improves with overriding.

f) Robust vs Related Metrics

Robustness is the degree to which a component performs correctly in the presence of invalid inputs or stressful environmental conditions.

i) Robust with WMC: WMC may not have impact on robustness.

ii) Robust with RFC: Variation in values of RFC may not change robustness.

iii) Robust with DIT: Robustness may not be affected by DIT values.

iv) Robust with CBO: Enriched coupling involves more complexity resulting into decrease in robustness with increasing CBO.

v) Robust with NOC: NOC may not influence robustness.

vi) Robust with AIF: Inheritance implies penetration into the class hierarchy having implications in a number of areas; hence AIF increase will reduce robustness.

vii) Robust with MHF: Encapsulation implies restricted access, more of specialisation and loss of flexibility. Such systems are sturdier. MHF increase makes the system more robust.

viii) Robust with PF: PF may not have an effect on robustness.
2. Maintainability Quality Factor

The various maintainability sub-factors are analyzability, changeability, stability, testability, correctability, adaptiveness, understandability, error debugging, reusability, modularity, extensibility and documentation as provided in Table 4.3. The maintainability sub-factors and related metrics are tabulated in Table 6.2.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Sub-factor</th>
<th>Related Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintainability</td>
<td>Analyzability</td>
<td>WMC, RFC, CBO, AIF, MHF</td>
</tr>
<tr>
<td></td>
<td>Changeability</td>
<td>WMC, RFC, CBO, AIF, MHF</td>
</tr>
<tr>
<td></td>
<td>Stability</td>
<td>RFC, CBO, AIF, MHF</td>
</tr>
<tr>
<td></td>
<td>Testability</td>
<td>WMC, RFC, CBO, AIF, MHF</td>
</tr>
<tr>
<td></td>
<td>Correctability</td>
<td>RFC, CBO, NOC, AIF, MHF, PF</td>
</tr>
<tr>
<td></td>
<td>Adaptiveness</td>
<td>RFC, CBO, NOC</td>
</tr>
<tr>
<td></td>
<td>Understandability</td>
<td>WMC, RFC, DIT, CBO, NOC, AIF, MHF, PF</td>
</tr>
<tr>
<td></td>
<td>Error Debugging</td>
<td>WMC, RFC, DIT, CBO, AIF, MHF, PF</td>
</tr>
<tr>
<td></td>
<td>Reusability</td>
<td>DIT, NOC, AIF, MHF, PF</td>
</tr>
<tr>
<td></td>
<td>Modularity</td>
<td>DIT, NOC, AIF</td>
</tr>
<tr>
<td></td>
<td>Extensibility</td>
<td>WMC, RFC, CBO, NOC, AIF, MHF, PF</td>
</tr>
<tr>
<td></td>
<td>Documentation</td>
<td>WMC, RFC, DIT, CBO, NOC, AIF, MHF, PF</td>
</tr>
</tbody>
</table>

a) Analyzability vs Related Metrics

Analyzability represents the efficiency with which the errors and deficiencies or the failures can be identified for a component. Increased code, responses, coupling and inheritance has adverse effect on analyzability.

i) Analyzability with WMC: Analyzability represents the capability of software product to be diagnosed for deficiencies or causes of failures in a class, or for identification of parts to be modified. Analyzability is directly related to the size and complexity of the class. Analyzability will decline with increasing WMC.
ii) **Analyzezability with RFC**: The number of executed methods in response to a message directly affect effort and time for analyzability. Higher value of RFC increases complexity. Analyzability shall decline with increasing RFC.

iii) **Analyzezability with DIT**: Values of DIT may not have affect on analyzability.

iv) **Analyzezability with CBO**: Analyzability would decrease with increasing CBO since the parts of the system are dependent on other parts of the system which need to be analyzed as well.

v) **Analyzezability with NOC**: NOC values may not have impinge on analyzability.

vi) **Analyzezability with AIF**: Attributes and methods available deep down in the classes' hierarchy are difficult to be analysed. Analyzability will decrease with increase in AIF.

vii) **Analyzezability with MHF**: Having limited or less access to methods and / or attributes helps in analysis as it restricts the domain area. Analyzability will increase with increase in MHF.

viii) **Analyzezability with PF**: Analyzability may not get affected by PF values.

b) **Changeability vs Related Metrics**

Changeability is the easiness of undertaking changes within a component. If the developer during the component development restricts the amount of source code, lessens the complexity, restricts the access to the methods and manages the bonding then the ability of change improves.
i) *Changeability with WMC:* Changing a class in a large complex system is difficult to handle as information regarding existing classes has to be known in advance. Higher value of WMC implies many classes hence, before changes are made many classes are to be analyzed. Changeability shall decline with increasing WMC.

ii) *Changeability with RFC:* Value of RFC means that as to how many methods are able to call the specific method. Higher value of RFC implies greater ability of method to be called from various methods. Each change must be correct for all execution paths. Thereby, changeability declines with increasing RFC.

iii) *Changeability with DIT:* Values of DIT may have no affect on changeability.

iv) *Changeability with CBO:* Changeability shall decrease with increasing CBO as the change in one part of system might involve the other parts which also need to be changed.

v) *Changeability with NOC:* Changeability may not get affected by NOC.

vi) *Changeability with AIF:* Even a minor change in an attribute may have wider impact in case of higher inheritance values. Increase in values of AIF shall result in the decline of changeability.

vii) *Changeability with MHF:* Higher values of encapsulation favour changeability. Increasing MHF values shall increase changeability.

viii) *Changeability with PF:* PF may have no influence on changeability.
c) Stability vs Related Metrics

Stability is the scarceness of risk of failures after a change in the software. The developer should restrict the use of responses, coupling and attributes being inherited to improve stability.

i) Stability with WMC: WMC may have no impact on stability.

ii) Stability with RFC: Due to reduced analyzability; stability may be influenced negatively by response of a class. Stability shall decline with increasing RFC.

iii) Stability with DIT: Stability is not affected by DIT.

iv) Stability with CBO: Stability correlates with the metrics which measure attributes of the software that indicate about the risk of unexpected effects as a result of modification. Stability decreases with increasing CBO since the effects of the modification may effect other parts as well.

v) Stability with NOC: Change in values of NOC may not influence stability.

vi) Stability with AIF: Attributes of a class available deep down in the hierarchy cause stability problems. Stability declines with an increase in AIF. Increase in AIF indicates more and more attributes are inherited with increasing AIF and stability will decline as the risk of unexpected effects as a result of modifications rise.

vii) Stability with MHF: Private methods indicate loss of flexibility and hence lesser modification ability. In the midst of increasing MHF stability too would increase.

viii) Stability with PF: Stability might not be affected by PF.
d) Testability vs Related Metrics

WMC is the measurement of complexity in OO domain. High values of WMC metric means a class with many complex methods. Therefore, WMC can be chosen as an indicator of maintainability since it affects fault proneness, stability and understandability. WMC is correlated with the effort needed to test that class. Since testability is a sub-character of maintainability so increase in WMC decreases testability.

i) Testability with WMC: Testability correlates with the metrics which measure the efforts needed for test coverage of all execution paths. The number of possible execution paths of a class increases with the number of methods and their control flow complexity. WMC allows an assessment of the number of methods and their complexity. Testability shall decline with increasing WMC.

ii) Testability with RFC: Complete testing requires exposure of all execution paths needed for test. Response For a Class computes the number of methods (directly) involved in handling a particular message. Testability will decline with increasing RFC.

iii) Testability with DIT: DIT may not influence testability.

iv) Testability with CBO: Testability would decrease with increasing CBO since one part of the system is linked to other part of the system which results in increased number of possible test paths.

v) Testability with NOC: Change in values of NOC might not affect testability.
vi) **Testability with AIF**: Increasing values of inheritance decreases testability. Growing AIF will diminish testability.

vii) **Testability with MHF**: Higher encapsulation assists testability in a way that rising values of M HF make the class less inheritable and thus reducing the test paths. M HF will help in raising testability.

viii) **Testability with PF**: Testability might not be influenced by PF.

e) **Correctability vs Related Metrics**

Effort required to locate and fix an error in a component between major releases is termed as correctability.

i) **Correctability with WMC**: WMC might not influence correctability.

ii) **Correctability with RFC**: Incorporating many responses increases complexity thus the developer has to consider a number of paths before fixing an error. Higher RFC reduces correctability.

iii) **Correctability with DIT**: Variation in values of DIT may not impinge on correctability.

iv) **Correctability with CBO**: Correctability shall decrease with increase in CBO. More the coupling more is the bonding between objects leading to the need of more care while correcting the component.

iv) **Correctability with NOC**: Correcting children which are designed for a specific task does not affect other modules, hence improving correctability with higher values of NOC.

vi) **Correctability with AIF**: Inheriting more attributes makes it worse for the developer to correct an error as the implications of the attribute are wide spread.
vii) Correctability with MHF: Hidden methods restrict access which means any change to the hidden method may not affect the other method. Increasing MHF improves correctability.

viii) Correctability with PF: Correcting an overridden class may affect in debugging the other class too. PF improves correctability.

f) Adaptiveness vs Related Metrics

It is the degree to which the component accepts the environment after the changes have been made.

i) Adaptiveness with WMC: Adaptiveness sub-factor may not change with change in values of WMC.

ii) Adaptiveness with RFC: Adaptiveness would decline with increasing RFC. The increase in the number of methods executed in response to message will also increase the number of methods to be changed for adaption.

iii) Adaptiveness with DIT: DIT value may not have impact on adaptiveness.

iv) Adaptiveness with CBO: Adaptiveness shall decrease with increasing CBO. Increase in CBO results in higher number of other classes that are coupled to a class hence making the changes difficult.

v) Adaptiveness with NOC: A class having a higher number of children is difficult to adapt to different specified environments. Adapting the parent class to a new environment, can make it unsuitable for the children, requiring adaptation in them as well. Adaptiveness will decrease with increasing NOC.
vi) **Adaptiveness with AIF**: Adaptiveness may not be affected by AIF.

vii) **Adaptiveness with MHF**: Variation in values of MHF may have no impact on adaptiveness.

viii) **Adaptiveness with PF**: Adaptiveness sub-factor may not be affected by change in values of PF.

**g) Understandability vs Related Metrics**

Understandability is the degree to which the purpose of the changes during maintenance is clearly understood.

i) **Understandability with WMC**: Understandability shall decline with increasing WMC. Increase in the WMC indicates increase in the sum of complexities of the methods of the class thus, reducing understandability.

ii) **Understandability with RFC**: Understandability will decline with increasing RFC as understandability of a class depends upon its complexity and response size of the method.

iii) **Understandability with DIT**: Classes that are deep down in the class hierarchy potentially inherit many methods from super-class. The increased DIT signifies greater design complexity thus reduced understandability.

iv) **Understandability with CBO**: Parts, which have a high coupling, may be highly inversely related to understandability, since they are using other parts of the system which need to be understood as well. Understandability would decrease with increasing CBO.

v) **Understandability with NOC**: Understanding a class is supported if it has a high number of children. Numerous children show directly
a high usability of the class under concern. Understandability shall increase with increasing NOC.

**vi)** Understandability with AIF: Increasing values of AIF would decrease understandability because increased reusability makes the component more complex and less understanding.

**vii)** Understandability with MHF: MHF increased values will also enable understandability to increase.

**viii)** Understandability with PF: It is difficult for the developer to understand the overridden classes. PF worsens understandability.

**h) Error Debugging vs Related Metrics**

Error debugging is the effort required to locate and fix an error in an operational program.

**i)** Error Debugging with WMC: Increased number of methods in a class makes it cumbersome to debug errors. Higher value of WMC lowers error debugging.

**ii)** Error Debugging with RFC: More responses mean more carefulness required by the developer while removing bugs. RFC decreases error debugging.

**iii)** Error Debugging with DIT: More DIT implies more difficulty in error debugging.

**iv)** Error Debugging with CBO: Increase in CBO lessens error debugging.

**v)** Error Debugging with NOC: Value of NOC might not affect error debugging.
vi) *Error Debugging with AIF*: Highly inherited attributes are prone to errors. Further, identifying such attributes is cumbersome. Increased AIF will reduce error debugging.

vii) *Error Debugging with MHF*: Increased MHF eases the process of error debugging as the highly encapsulated classes are less complex and easy to debug.

viii) *Error Debugging with PF*: Removal of bug in an overridden class rectifies the other class too. Thereby, PF improves error debugging.

i) **Reusability vs Related Metrics**

Reusability is the ease with which an existing application or component can be reused. It is the extent to which a component can be used with other components related to the packaging and scope of the functions that programs perform.

i) *Reusability with WMC*: WMC may not affect reusability.

ii) *Reusability with RFC*: Variation in values of RFC might not influence reusability.

iii) *Reusability with DIT*: It is positively influenced by DIT. A large DIT value indicates that many methods might be reused.

iv) *Reusability with CBO*: Changing coupling effect may not have impact on reusability.

v) *Reusability with NOC*: It is positively influenced by attributes assessed with NOC as higher NOC denotes higher use of the base class.

vi) *Reusability with AIF*: Inheritance promotes the reuse of attributes and methods. Increase in AIF will result in increase in reusability.
vii) **Reusability with MHF:** MHF negatively influence reusability since high MHF indicate very little functionality and the methods are not available for reuse.

viii) **Reusability with PF:** While the component is being developed overriding helps in faster and better making of classes. Thereby, PF improves reusability.

**j) Modularity vs Related Metrics**

Modularity is the functional independence of program components. It is increased when it is possible to divide each component into sub-components.

i) **Modularity with WMC:** Change in values of WMC may not change modularity.

ii) **Modularity with RFC:** Modularity may not be affected by RFC.

iii) **Modularity with DIT:** As access is made deeper it implies that modules are directly or indirectly formulated. With increase in DIT, modularity will increase.

iv) **Modularity with CBO:** Variation in assessment of CBO may not affect modularity.

v) **Modularity with NOC:** Higher values of NOC would enable modules to be formed. Thereby, NOC and Modularity are positively related.

vi) **Modularity with AIF:** Inheritance favours modularity either in direct manner or indirectly. Growing of AIF develops modularity too.

vii) **Modularity with MHF:** MHF may have no impact on modularity.

viii) **Modularity with PF:** Modularity might not be affected by values of PF.
k) Extensibility vs Related Metrics

Extensibility is the ease with which a component can be enhanced in the future to meet changing requirements or objectives.

i) Extensibility with WMC: WMC is a measure of complexities; increased levels of complexities will be a hindrance for the component to be augmented. Increased value of WMC worsens extensibility.

ii) Extensibility with RFC: If communication among classes exists on a larger scale then it requires more effort for adding new components. Increase in RFC would decline extensibility.

iii) Extensibility with DIT: DIT may not have any influence on the ability to enhance a component.

iv) Extensibility with CBO: To provide an extendable program it is a good practice to reduce the dependencies between implementing classes. With an increase in CBO, extensibility will decrease.

v) Extensibility with NOC: Incrementing size of component by adding children or modules are acceptable. Developer shall be at ease to join independent children as it makes the process more manageable. NOC improves extensibility.

vi) Extensibility with AIF: While enhancing software by adding new features, it is needed to closely monitor the adverse effect of the new feature on the existing components. Component having minimum inheritance levels will be less affected by the addition of a new feature. More AIF will denote less extensibility.
vii) **Extensibility with MHF:** MHF is directly related to extensibility. Inaccessible objects from one another would lead to confinement in their scope. Addition of a new component, in a limited scope zone produces fewer hassles.

viii) **Extensibility with PF:** Already overridden class if further is to be extended it shall lead to complexity. PF reduces extensibility.

1) **Documentation vs Related Metrics**

Documentation is the degree to which the source code provides meaningful documentation. After the modifications have been made the changes have to be documented.

i) **Documentation with WMC:** Increase in WMC means more efforts to understand which signifies more need of documentation, but difficult to document.

ii) **Documentation with RFC:** Complexity increases with increase in RFC. Implying thereof that with increase in RFC documentation need grows. However, documenting a complex component is difficult.

iii) **Documentation with DIT:** Higher values of DIT may affect documentation sub-factor in an inverse manner. Deeper classes are difficult to explain, maintain and document.

iv) **Documentation with CBO:** The coupled objects are difficult to be documented, especially after modification of the component. CBO reduces documentation.

v) **Documentation with NOC:** Independent modules in the form of children are easily documented. NOC improves documentation.
vi) **Documentation with AIF:** More inheritance requires more description of attributes and methods. Increase in AIF will give rise to documentation needs.

vii) **Documentation with MHF:** Hidden classes do not interfere with other classes etc. hence, are easily documented. MHF improves documentation.

viii) **Documentation with PF:** Classes using the feature of polymorphism are difficult to be documented. PF reduces documentation.

### 3. Portability Quality Factor

The various portability sub-characteristics are: adaptiveness, replaceability, reusability and transferability as depicted in Table 4.4. The sub-factors of portability and related metrics are tabulated in Table 6.3.

#### TABLE 6.3: SUB-FACTORS OF PORTABILITY AND RELATED METRICS

<table>
<thead>
<tr>
<th>Factor</th>
<th>Sub-factor</th>
<th>Related Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>RFC, CBO, NOC</td>
</tr>
<tr>
<td></td>
<td>Replaceability</td>
<td>RFC, DIT, CBO, NOC, AIF, MHF</td>
</tr>
<tr>
<td></td>
<td>Reusability</td>
<td>DIT, NOC, AIF, MHF, PF</td>
</tr>
<tr>
<td></td>
<td>Transferability</td>
<td>WMC, RFC, DIT, CBO, NOC, AIF, MHF</td>
</tr>
</tbody>
</table>

#### a) Adaptability vs Related Metrics

The ability of the software to adjust according to the changed environment is termed as adaptability. To make smooth transition from one environment to another after changes have been made to the component, the developer of the component has to place efforts.

i) **Adaptability with WMC:** Deviation in values of WMC might not influence adaptability.
ii) Adaptability with RFC: Component having high level of communication among classes has less scope for adaptability. RFC worsens adaptability.

iii) Adaptability with DIT: Adaptability may not be changed due to change in values of DIT.

iv) Adaptability with CBO: Highly coupled components face tough task in adapting to an environment especially when the component or its part has changed. CBO declines adaptability.

v) Adaptability with NOC: Children constitute of independent modules or self-sufficient parts of components, the developer has to place more efforts to make each modified child adaptable to the environment. NOC weakens adaptability.

vi) Adaptability with AIF: AIF may not cause adaptability to change.

vii) Adaptability with MHF: Adaptability might not be changed due to changing values of MHF.

viii) Adaptability with PF: PF may not affect adaptability.

b) Replaceability vs Related Metrics

While transporting a component, the ability of the component to be used instead of the existing component is called replaceability.

i) Replaceability with WMC: Variation in values of WMC may have no impact on replaceability.

ii) Replaceability with RFC: Replaceability will decline with increasing RFC, because components with complex response sets and control structures might have complex apparent behavior making it more
difficult to check substitutability and to substitute a component in reality.

iii) Replaceability with DIT: The substitute of a component must imitate its interface. Large interfaces are difficult to check for substitutability and to actually substitute. Interface size increases for classes that are deep down in the class hierarchy. Replaceability will decline with increasing DIT.

iv) Replaceability with CBO: Coupling between objects raises issues pertaining to complexity. Uncoupled objects are easier to augment than those with high degree dependencies due to lower interactions and interconnections. Higher CBO shall reduce replaceability.

v) Replaceability with NOC: A class having independent children provides an opportunity to the developer to coin a class and replace it with an existing class without affecting other segments of the code. Replaceability improves with increasing NOC.

vi) Replaceability with AIF: Replacing a method or an attribute with a substitute deep down in the class hierarchy involves more effort. Replaceability would decline with increasing AIF.

vii) Replaceability with MHF: Substituting in a limited domain area is manageable. It implies that higher values of MHF restrict the methods accessibility making it easier for replaceability to increase.

viii) Replaceability with PF: Change in values of PF may have no impact replaceability.
c) **Reusability vs Related Metrics**

Use of the component more than once in different domain area is related to reusability. Higher degree of reusability reduces the effort required to create another piece of code.

i) *Reusability with WMC*: WMC may have no affect on reusability.

ii) *Reusability with RFC*: Reusability might not be affected by change in value of RFC.

iii) *Reusability with DIT*: Classes available deep down in the hierarchy tree have better scope of being reused. DIT improves reusability.

iv) *Reusability with CBO*: Coupling may not affect reusability.

v) *Reusability with NOC*: Chances of independent classes being reused are more. The independent child may be incorporated in various components. NOC enhances reusability.

vi) *Reusability with AIF*: Inherited attributes have better chance to be reused again. AIF increases reusability.

vii) *Reusability with MHF*: Higher value of MHF denotes that methods are hidden. Such methods may not be reused thus decreasing reusability.

viii) *Reusability with PF*: Source code of overriding classes is used again in any case. Thereby, PF increases reusability.

d) **Transferability vs Related Metrics**

Transferability measures the degree of effort to be placed in moving a component from one to another environment.

i) *Transferability with WMC*: Handling more methods require more efforts which will decline transferability with increasing WMC.
ii) Transferability with RFC: A lot of communication makes the component more complex thereby would decline transferability with increasing response set size (RFC).

iii) Transferability with DIT: Deeper class hierarchy based systems are more organised and are better placed for transferability. Transferability will increase with increasing DIT.

iv) Transferability with CBO: More coupling will decrease the ability to transfer, due to increased complexity.

v) Transferability with NOC: Breadth wise spread classes within a component require more efforts to coordinate among them. Transferability is negatively influenced by attributes assessed with NOC.

vi) Transferability with AIF: Inherited classes within a component are more structured. Transferability will increase, with increasing value of AIF.

vii) Transferability with MHF: Encapsulation leads to data abstraction. Hidden methods can easily be ported thus the ability to transfer will increase with increase in MHF.

viii) Transferability with PF: Variation in values of PF may not influence transferability.

4. Reliability Quality Factor

The quality sub-factors for reliability are: maturity, fault tolerance, recoverability, survivability, error tolerance, integrity and simplicity. The related metrics of reliability are tabulated in Table 6.4.
TABLE 6.4: SUB-FACTORS OF RELIABILITY AND RELATED METRICS

<table>
<thead>
<tr>
<th>Factor</th>
<th>Sub-factor</th>
<th>Related Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>Maturity</td>
<td>WMC, RFC, DIT, CBO, AIF, MHF</td>
</tr>
<tr>
<td>Reliability</td>
<td>Fault Tolerance</td>
<td>WMC, RFC, DIT, CBO, NOC, AIF, MHF</td>
</tr>
<tr>
<td>Reliability</td>
<td>Recoverability</td>
<td>WMC, DIT, CBO, AIF, MHF, PF</td>
</tr>
<tr>
<td>Reliability</td>
<td>Survivability</td>
<td>WMC, RFC, DIT, CBO, NOC, AIF, MHF</td>
</tr>
<tr>
<td>Reliability</td>
<td>Error Tolerance</td>
<td>WMC, RFC, DIT, CBO, NOC, AIF, MHF</td>
</tr>
<tr>
<td>Reliability</td>
<td>Integrity</td>
<td>RFC, CBO, NOC, AIF, MHF</td>
</tr>
<tr>
<td>Reliability</td>
<td>Simplicity</td>
<td>WMC, RFC, DIT, CBO, NOC, AIF</td>
</tr>
</tbody>
</table>

a) Maturity vs Related Metrics

Maturity is the degree to which a component avoids failure due to presence of bugs in the component.

i) Maturity with WMC: Maturity correlates with metrics which measure the frequency of failure due to faults in the component. Maturity will decline with increasing WMC. It is due to reduced testability as bugs might be left in the component.

ii) Maturity with RFC: Maturity might decline with increasing RFC since large response set size reduces testability thus making the system more bug prone.

iii) Maturity with DIT: Increased DIT suggests reduced testability implying bugs might be left in the software. Therefore, maturity might decline with increasing DIT.

iv) Maturity with CBO: Lot of coupling between objects leads to complexity and the after effects of such a component are unpredictable and may cause system to fail. CBO worsens maturity.

v) Maturity with NOC: Variation in value of NOC may not affect maturity sub-factor.
vi) Maturity with AIF: Maturity diminishes with increase in AIF as it is difficult to keep a track of increasing inheritance, which worsens the ability to test.

vii) Maturity with MHF: Maturity will increase with increase in MHF since higher value of MHF shows less complexity and reduced bug density.

viii) Maturity with PF: PF may not affect maturity.

b) Fault-tolerance vs Related Metrics

Fault tolerance is the ability to recover itself without disrupting the services.

i) Fault-tolerance with WMC: WMC is the sum of complexities. With an increase in WMC the complexity increases thus decreasing the chance of recovering the component. Fault-tolerance decreases if WMC increases.

ii) Fault-tolerance with RFC: Fault-tolerance decreases with an increase in RFC. Due to increased responses the complexity increases resulting in less chance of fault-tolerance.

iii) Fault-tolerance with DIT: Relating DIT to fault-tolerance requires being able to locate tree hierarchy of a system responsible for interoperability. A high DIT in these hierarchy structures might indicate a better ability to interact. Fault-Tolerance might increase with increasing DIT.

iv) Fault-tolerance with CBO: Increase in CBO indicate high coupling between the different parts of the system and can be affected by
faults in other parts of the system. Fault-Tolerance might decrease with increasing CBO.

v) Fault-tolerance with NOC: In case of independent children failing due to bug still the software may function properly with the remaining features. NOC improves fault-tolerance.

vi) Fault-tolerance with AIF: Deeper inheritance levels imply more complexity it means the component is less fault tolerant. Higher value of AIF will mean less fault tolerance.

vii) Fault-tolerance with MHF: Increase in MHF results in decrease in defect density and effort to fix defects. Increasing MHF value will result in more fault tolerance.

viii) Fault-tolerance with PF: Change in values of PF may not influence fault tolerance.

c) Recoverability vs Related Metrics

Recoverability implies to restore to a normal state after a failure or crash.

i) Recoverability with WMC: Complex component is more difficult to restore Higher values of WMC might indicate a higher complexity. Recoverability might decrease if the WMC increases.

ii) Recoverability with RFC: RFC may not affect recoverability.

iii) Recoverability with DIT: Relating DIT to recoverability requires being able to locate hierarchy structures of a system responsible for recovery. A high DIT in these hierarchy structures might indicate a higher recoverability. Recoverability might increase with increasing DIT.
iv) Recoverability with CBO: Sections that have a high (outgoing) efferent coupling may be inversely related to recoverability, since their data is distributed in the entire system making recovery difficult. Recoverability might decrease with increasing CBO.

v) Recoverability with NOC: Change in values of NOC may not have any impact on recoverability.

vi) Recoverability with AIF: Inheritance enables access to its methods and attributes. Inherited objects will have a vast implication and dependence increases, which may lead to break down of many objects. Increase in values of AIF will decrease the chances of recoverability.

vii) Recoverability with MHF: Encapsulation leads to data hiding and MHF hide methods. Higher values of MHF denote that the concerned method and/or attribute are created for a specific cause only. Its working or malfunctioning will block only a limited number of methods / attributes. The remaining ones shall work normally. Thereby, increase in MHF will increase recoverability.

viii) Recoverability with PF: Recovering from one of the overridden class affects in making progress from another too. PF improves recoverability.

d) Survivability vs Related Metrics

Survivability is the ability to deliver results in the presence of faults and threats to the component.
i) *Survivability with WMC*: WMC directly correlates with the metrics measuring complexity. Survivability will decrease with increase in WMC.

ii) *Survivability with RFC*: More responses imply more complexity and lesser scope of surviving.

iii) *Survivability with DIT*: Increase in DIT will decrease survivability.

iv) *Survivability with CBO*: CBO will reduce survivability. It can be due to the fact that if one part of the system is affected by malicious harm then it can affect other parts of the system due to high coupling.

v) *Survivability with NOC*: NOC implies independent children performing independent tasks. A break down in one of the child classes may not affect other classes. Presence of high degree of NOC improves survivability.

vi) *Survivability with AIF*: Increase in inheritance levels makes the system more vulnerable to problems. Increase in AIF will decrease survivability.

vii) *Survivability with MHF*: Increase in encapsulation results in a secure system. A secure system will be less prone to malicious harm. Hence, increase in MHF will increase survivability.

viii) *Survivability with PF*: PF may not affect survivability.

e) **Error-tolerance vs Related Metrics**

The extent to which the component performs normally after accepting erroneous input is called error tolerance.
i) *Error-tolerance with WMC*: WMC is a measure of complexity; more methods per class imply more complexity and less error-tolerant.

ii) *Error-tolerance with RFC*: Error-tolerant components need to be less complicated whereas RFC increases the complexity. Increase in RFC reduces error-tolerance.

iii) *Error-tolerance with DIT*: Relating DIT to error-tolerance requires being able to locate tree hierarchy of a system responsible for interoperability. A high DIT in these hierarchy structures might indicate a better ability to interact. Error-Tolerance might increase with increasing DIT.

iv) *Error-tolerance with CBO*: Increase in CBO indicate high coupling between the different parts of the system and can be affected by errors in other parts of the system. Error-Tolerance might decrease with increasing CBO.

v) *Error-tolerance with NOC*: More children imply better chance for a component to manage the functioning as each child caters to the needs of a specialised task only. NOC improve error-tolerance.

vi) *Error-tolerance with AIF*: Deeper inheritance levels imply more complexity it means the component is less error tolerant. Higher value of AIF would mean less error tolerance.

vii) *Error-tolerance with MHF*: Increase in MHF results in decrease in defect density and effort to fix defects. Increasing MHF value will result in more error tolerance.

viii) *Error-tolerance with PF*: Variation in values of PF may not affect error tolerance.
f) **Integrity vs Related Metrics**

Integrity is the extent to which a component restricts unauthorised access.

i) **Integrity with WMC**: WMC may have no impact on integrity.

ii) **Integrity with RFC**: More number of responses to a class dilutes the affect of restricting access and weakens integrity. RFC declines integrity.

iii) **Integrity with DIT**: Integrity may not be affected by values of DIT.

iv) **Integrity with CBO**: Coupling involves interaction among classes which involves a decrease in integrity.

v) **Integrity with NOC**: Individual classes do not affect other classes. NOC restores integrity.

vi) **Integrity with AIF**: Growing inheritance in the form of AIF will increase accessibility which in a way decreases integrity.

vii) **Integrity with MHF**: Rising values of MHF improve integrity. Binding of scope facilitate in increasing integrity.

viii) **Integrity with PF**: Variation in values of PF may not affect integrity.

g) **Simplicity vs Related Metrics**

Simplicity is involved with the ease with which the component is understood for all purposes.

i) **Simplicity with WMC**: Managing bigger size is more complex than smaller ones. Size is weighed by the weighted number of classes. Simplicity will decrease with an increase in WMC.
ii) *Simplicity with RFC*: More communication within classes and between classes lead to complexity. Increase in RFC will be inversely related to simplicity.

iii) *Simplicity with DIT*: Deeper a class within the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behaviour, depth in the class hierarchy implies methods are being inherited from super classes. Simplicity shall decrease with increasing DIT.

iv) *Simplicity with CBO*: Coupling leads to more dependence on other objects. Simplicity will decrease with an increase in CBO.

v) *Simplicity with NOC*: Increase in NOC shows the way for small manageable units. Simplicity will increase with increase in NOC.

vi) *Simplicity with AIF*: Inherited methods and attributes have the ability to make an impact on a wider front; consequently more care is needed. Simplicity would decrease with increasing values of AIF.

vii) *Simplicity with MHF*: Impact of hidden methods on simplicity may not exist.

viii) *Simplicity with PF*: PF may not affect simplicity.

5. **Usability Quality Factor**

The various sub-factors for usability quality factor are documented in Table 4.6 and are namely, attractiveness, understandability, learnability, operability and documentation. The sub-factors of usability and related metrics are tabulated in Table 6.5.
### TABLE 6.5: SUB-FACTORS OF USABILITY AND RELATED METRICS

<table>
<thead>
<tr>
<th>Factor</th>
<th>Sub-factor</th>
<th>Related Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usability</td>
<td>Attractiveness</td>
<td>WMC, RFC, DIT, CBO, NOC, AIF, MHF</td>
</tr>
<tr>
<td></td>
<td>Understandability</td>
<td>WMC, RFC, DIT, CBO, NOC, AIF, MHF, PF</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
<td>WMC, RFC, DIT, CBO, NOC, AIF, MHF, PF</td>
</tr>
<tr>
<td></td>
<td>Operability</td>
<td>WMC, RFC, DIT, CBO, NOC</td>
</tr>
<tr>
<td></td>
<td>Documentation</td>
<td>WMC, RFC, DIT, CBO, NOC, AIF, MHF, PF</td>
</tr>
</tbody>
</table>

---

**a) Attractiveness vs Related Metrics**

Attractiveness is referred as user interface aesthetics and is an indicator expressing the extent of user liking the component during the operation. To make the component attractive the developer incorporates various means and mechanisms.

1. **Attractiveness with WMC:** Attractiveness of a class depends on the maximum availability of; features, input data mechanisms, output information means, etc. Adding these factors increases the source code size of the component and the methods per class, thus increasing WMC. Attractiveness may get increased with increasing WMC.

2. **Attractiveness with RFC:** A class able to respond from multiple approaches is likely to be more attractive for the user. Thereby, RFC may improve attractiveness of a class.

3. **Attractiveness with DIT:** Attractiveness of a class depends on the size of the potentially reused code. Classes that are deep down in the class hierarchy potentially inherit many methods from superclasses. Attractiveness will increase with increasing DIT.

4. **Attractiveness with CBO:** Parts that have a high (outgoing) coupling may be highly inversely related to attractiveness, since they are using other parts of the system which need to be understood as
Mapping Quality Factors With Metrics And Quality Assurance Framework

well, and represent dependencies. Attractiveness shall decrease with increasing CBO.

v) **Attractiveness with NOC:** A class having a higher number of children appears more attractive to the software engineer / developer. In a class diagram it is an eye-catcher because it is highlighted by having many children. Linked assumptions could be, that the class is steady, since it is tested each time a child is tested, that is well documented and understood, since it has been extended, that it is easier to understand, since there are many examples of usage, that it helps to understand the children, if the parent class has been understood, that is plays a central role in the design, since many classes extend its functionality. Attractiveness will increase with increasing NOC.

vi) **Attractiveness with AIF:** Inheritance promotes reusability as well as usability. Attractiveness shall increase with increasing values of AIF.

vii) **Attractiveness with MHF:** Values of MHF restrict reuse as accessibility is confined. Increasing values of MHF would decline attractiveness.

viii) **Attractiveness with PF:** Values of PF may have no influence on attractiveness.

b) **Understandability vs Related Metrics**

Understandability is the perceiveness of the complete function and component.
i) **Understandability with WMC**: Increase in WMC increases complexity and deteriorates the power of understanding.

ii) **Understandability with RFC**: Existence of a number of responses imply more applying of concentration by developer and increases complexity. Thus, RFC reduces understandability.

iii) **Understandability with DIT**: Classes existing deep down in the inheritance tree are difficult to be understood. DIT worsens understandability.

iv) **Understandability with CBO**: Numerous tightly coupled objects increase complexity and reduce understandability.

v) **Understandability with NOC**: NOC involves modularity and setting of independent scope of tasks. Higher value of NOC makes understandability better.

vi) **Understandability with AIF**: To understand inherited attributes more effort is required. Understandability drops with rising AIF.

vii) **Understandability with MHF**: Hidden methods provide limited access. Such methods are easy to understand. MHF improves understandability.

viii) **Understandability with PF**: Overridden classes are difficult to be understood hence, PF reduces understandability.

**C) Learnability vs Related Metrics**

Learnability is the ease of learning, ease of using help system and effectiveness of help system.

i) **Learnability with WMC**: Learnability is correlated to the measure of the user’s effort to learn an application. Learnability of a class
depends upon the size of its interfaces. Learnability would decline with increasing WMC.

ii) Learnability with RFC: Learnability shall decline with increasing RFC as learnability depends on the complexity of its interface and the number of methods in other classes called in response to a received message.

iii) Learnability with DIT: As DIT becomes high it makes the classes harder to analyze and in turn to learn. Learnability will decline with increasing DIT.

iv) Learnability with CBO: Learnability would decrease with increasing CBO because due to high CBO some parts of the system may have high coupling with other parts which also needs to be understood.

v) Learnability with NOC: Learnability will increase with increasing NOC. High NOC denotes high number of children which require versatile knowledge on how to reuse a particular class in different situations.

vi) Learnability with AIF: Learnability shall decline with increasing AIF. Lengthier class hierarchy leads to keeping the knowledge of various inherited attributes hence making the learnability difficult.

vii) Learnability with MHF: Restricting the scope of a method in its definition makes understanding easy. Learnability will increase with increase in MHF.

viii) Learnability with PF: It is difficult to learn and use overridden classes therefore PF reduces learnability.
d) Operability vs Related Metrics

Operability is the ability to provide self explanatory error messages, correction of errors, ability to undo the actions taken and able to customize the operations.

i) Operability with WMC: Operability shall decline with increasing WMC. It is due to the fact that increased WMC shows higher complexity and thus reduced operability.

ii) Operability with RFC: Operability intends to keep many options ready in terms of responses and ability to perform by number of ways. As RFC increases operability increases.

iii) Operability with DIT: Increase in the DIT will reduce operability. The definitions of inherited methods are not local to the class which makes a class harder to analyse.

iv) Operability with CBO: Operability might decrease with increasing CBO as higher CBO represents higher dependency between the different parts of the system.

v) Operability with NOC: A high number of children indicate that a particular class is well integrated into an existing software system. This means, that it is suitable for several different tasks, self-descriptive since there exist many examples on its usage, having a higher error tolerance, since it is involved each time one of its children is tested. Operability will increase with increasing NOC.

vi) Operability with AIF: AIF may not affect operability.

vii) Operability with MHF: Variation in values of MHF may have no impact on operability.
viii) Operability with PF: Operability may not be influenced by PF.

e) **Documentation vs Related Metrics**

Documentation is the process of recording actions to be performed to get the task accomplished it is presented in the form of a manual.

i) *Documentation with WMC*: Though it is necessary to document a complex component but it is difficult to do so. WMC complicates the process of documentation.

ii) *Documentation with RFC*: Increasing instances of responses for a class makes documentation difficult to implement.

iii) *Documentation with DIT*: It is all the more difficult to explain classes available deep down in the inheritance tree. DIT reduces documentation.

iv) *Documentation with CBO*: A number of bonding of the classes increases complexity which is complicated to describe. CBO reduces documentation.

v) *Documentation with NOC*: Increase in the values of NOC makes the component more systematic and easy to interpret. NOC helps in documentation.

vi) *Documentation with AIF*: Attributes available in inherited form are not easy to interpret. AIF hinders documentation.

vii) *Documentation with MHF*: Hidden methods have limited access and are confined to limited scope such methods are easy to explain. MHF assists in documentation.
viii) Documentation with PF: Overridden methods are difficult to elucidate. PF hampers documentation.

6.2 Quality Assurance Framework

A quality assurance framework is the set of factors, sub-factors and the relationships with the quality metric, which provide the basis for evaluating quality. A framework for software quality assurance is presented for CBSD. A common approach for formulating a model for software quality is to first identify a set of high level quality factors. The selected high level quality factors pertaining to the study are efficiency, maintainability, portability, reliability and usability. Then in a top down fashion these factors are decomposed into a set of sub-factors e.g. in this study efficiency is decomposed into sub-factors namely time behaviour, resource behaviour, reply time, processing speed, execution efficiency, robust, hardware independence and compliance. Similarly sub-factors for the other quality factors were identified. The various component quality metrics were identified in the next step. A quality metric provides a way to measure and quantify a quality sub-factor. The metrics suites selected were CK metrics suite and MOOD metrics suite. The process is followed by validating the identified metrics by applying them on to the components. After the validation, an investigation was made to determine the correlation between the software quality sub-factors and metrics from the developer's perspective. Based on the mapping between quality sub-factors and the metrics the quality assurance framework is proposed as presented in Table 6.6.
### TABLE 6.6: SOFTWARE QUALITY ASSURANCE FRAMEWORK

| Factors  | Metrics | W | M | C | R | F | C | D | I | T | B | O | C | R | F | C | D | I | T | C | C | A | I | M | H | P | F |
| Time Behavior          | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Resource Behavior      | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Reply Time             | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Processing Speed       | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Execution Efficiency   | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Robust                | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Analyzability          | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Changeability          | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Stability              | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Testability            | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Correctability         | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Adaptiveability        | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Understandability      | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Error Debugging        | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Reusability            | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Modularity             | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Extensibility          | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Documentation          | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Adaptability           | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Replaceability         | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Reusability            | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Transferability        | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Maturity               | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Fault Tolerance        | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Recoverability         | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Survivability          | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Error Tolerance        | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Integrity              | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Simplicity             | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Attractiveness         | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Understandability     | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Learnability           | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Operability            | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Documentation          | ▼ ▼ ▼ ▼ ▼ ▼ ▼ | | | | | | | | | | | | | | | | | | | | | | | | | | | |

where ▼: quality sub-factor decreases with the increasing metric value
▼: quality sub-factor increases with the increasing metric value
○: quality sub-factor is not effected by increasing or decreasing metric value
Mapping Quality Factors With Metrics And Quality Assurance Framework

The metrics are tabulated on the top row, the quality factors being put on the first column, with sub-factors placed next to the factor's column.

In the framework symbols are used to designate the relationship between the metrics and sub-factors. An upside arrow symbol (↑) depicts a direct positive relation between the metrics and the software quality sub-factor. A direct positive relation indicates that if the metric value increases so does the quality sub-factor. Similarly, a down side arrow symbol (↓) signifies an inverse relation between the metrics and the software quality sub-factor. The inverse relation means that if there is an increase in the metric value, then the quality sub-factor will show a decrease and vice versa. The symbol (⊙) is used to signify that no relation between the metrics and the quality sub-factor exists. The results obtained and further analysis has concluded that for certain relationships between sub-factors and metrics no relation exists. Incline or decline of the metrics value has insignificant effect on the software metric. The study was confined to evaluation of software quality keeping in view the developer's perspective. Evaluation of the relations between 34 software quality factors with 8 software quality metrics formed the foundation of the framework. These relationships were derived based on the rational analysis of the factors and the sub-factors. Depending upon the relationships it can be derived that efficiency can be measured from the metrics associated with the sub-factors listed for efficiency quality factor in the allusion Table 6.1. As depicted in Table 6.6 the sub-factor time
behaviour of quality factor efficiency may have a direct relation with MHF and PF. Further, time behaviour may have inverse relationship with WMC, RFC, CBO and AIF. Time behaviour may not have relation with DIT and NOC. The same analogy may be applied in interpreting the quality assurance framework. Thereby, the quality sub-factors may be controlled and monitored by varying the values of computable metrics and thus, eventually the quality of the software component may be enhanced. The software metrics are measurable and meticulously definable at all stages of component development. Therefore, the quality of the component is traceable, controllable and may be improved throughout the component development.

6.3 Conclusions

Mapping of metrics with quality factors and with quality sub-factors was accomplished. The study framed a software quality assurance model comprising of 8 metrics, 5 quality factors and 34 quality sub-factors. Subsequently, the quality assurance framework featured the type of relationship between the metrics and quality sub-factors, was proposed. The framework is of benefit to the software developer as it facilitates the developer in augmenting the quality of the software component on the basis of the values of the software metrics. The metrics are computable, precise and are easily interpretable in all the phases of development. The relationship depicted in the quality assurance framework between the metrics and the quality sub-factors assist in improving the quality of the software component.