Improving Software through Metrics while Providing Cradle to Grave Support

Jennifer Walters, Northrop Grumman
Kevin MacG. Adams, Ph.D., NCSOSE

Abstract. Metrics are beneficial to an organization that supports a product from inception through product retirement and disposal. Quality metrics have a critical role in this type of environment because they span both the development and operations and maintenance phases of the software life cycle, and there is a relationship between the internal quality metrics collected during development and the external quality metrics collected once the product is deployed. The key finding is that internal metrics can be collected early in the software development phase to predict the support required during the operations and maintenance phase; likewise, external metrics can be collected to drive software development process improvements. Finally, analyzing the relationships between the two can drive overall process improvements for the entire software lifecycle.

Role of Metrics
It is important to understand the role metrics play in the overall software lifecycle. First, the metrics of concern in this paper are quality attributes because maintenance efforts are highly dependent on the overall quality of the software [3]. In addition, quality attributes span both pre-delivery and post-delivery phases of the lifecycle and are therefore specifically relevant when a single organization supports the software over the entire lifecycle. Second, quality attributes are categorized as either internal or external [4]. Internal quality attributes are those that can be directly measured purely on the basis of product features such as size, length, or complexity [5]. External metrics are measurements that are dependent on how the software interacts with its environment and can therefore only be collected after the product has been deployed and operated during the maintenance and operations phase of the software lifecycle [5]. The remainder of this discussion involves the relationship between internal and external metrics.

Introduction
Software development is “the specification, construction, testing and delivery of a new application or of a discrete addition to an existing application” [1] while a maintenance project is “a software development project described as maintenance to correct errors in an original requirements specification, to adapt a system to a new environment, or to enhance a system” [1]. Often times these two processes are supported by two different organizations with two different goals. One team will focus on building the initial application and their primary concern is building a product that fulfills the requirements within both cost and schedule constraints. The second team has the responsibility of supporting the product during the remainder of the lifetime and has a primary concern of maintainability. As a result, the maintenance team is at the mercy of the software design and processes imposed by the software development team. While the resulting product might meet all the user requirements and appear to be a superb product, it is likely that the software development team did not build the initial product with maintenance in mind [2]. This results in a product that is more difficult and costly to maintain.

One Organization Supporting Entire Lifecycle
When the same organization supports both the development phase and the operations and maintenance phase, however, there are some opportunities to create a synergy between development and maintenance efforts. The focus can shift from individual phases to the overall software lifecycle. By transitioning focus, team members can collect measurements early in the development phase that can help predict issues in maintenance. Likewise, maintenance related metrics can be analyzed to provide guidance for improving development processes. Cause and effect analysis is a very powerful technique in this situation.

Chidamber and Kemerer Metrics
Chidamber and Kemerer (CK) metrics can “assist users in understanding object oriented design complexity and in forecasting external software qualities for example software defects, testing, and maintenance effort” [13]. Numerous research studies have validated CK metrics as a method for predicting maintainability [8, 13-16]. The suite includes six metrics: (1) weighted methods per class (WMC), (2) depth of inheritance tree (DIT), (3) number of children (NOC), (4) coupling between objects/classes (CBO), (5) response for a class (RFC), and (6) lack of cohesion in method (LCOM). Each of the six measurements in the CK suite quantify different quality attributes which relate to maintainability qualities; however, the last metric discussed in the next section, LCOM, has been shown to have the greatest impact on the total number of defects [8].

Weighted Methods per Class
Weighted Methods per Class (WMC) is a complexity measurement. However, Chidamber and Kemerer did not propose
a definition or method for measuring complexity. “If methods complexities are considered to be unity, the WMC metric turns in to the number of methods in a class” [17]. Whether it is general-ized to a count of methods or is more specialized through the use of a complexity algorithm, a higher WMC indicates a class that is more difficult to understand and modify.

**Depth of Inheritance Tree**

Depth of Inheritance Tree (DIT) is object oriented (OO) specific-ic as it measures the OO characteristic inheritance. Inheritance is “a semantic notion by which the responsibilities (properties and constraints) of a subclass are considered to include the responsibilities of a superclass, in addition to its own, specifically declared responsibilities” [1]. It is often described as an isa relationship. For example, a cat isa mammal. This can be extended to include a mammal isa animal. DIT measures the hierarchy of inheritance. In this example, there is a hierarchy of three: cat > mammal > animal. The deeper a class is in the inheritance tree, the more difficult it is to comprehend and predict the behavior of the class [18].

**Number of Children**

Number of Children (NOC) is similar to DIT as it is related to inheritance. It is the count of immediate sub-classes in the class hierarchy [13]. NOC takes a more horizontal approach to inheritance. Instead of walking down the inheritance tree, it counts the number of classes inheriting methods from the parent class. Extending the previous example, a reptile is also an animal. Now both mammal and reptile classes inherit the members from animal. So, if the animal base class is modified there is potential impact to both sub-classes. High NOC measurements require more impact analysis.

**Coupling Between Objects Classes**

As the name suggests, Coupling Between Objects Classes (CBO) measures coupling. Coupling is “the manner and degree of interdependence between software modules” [1]. A class is considered coupled with another class if its members are used by another class. Coupling makes it more difficult to isolate units of code for testing. The interdependence also makes code comprehension more difficult and increases the need for impact analysis. It is “highly connected to portability, maintainability, and re-usability” [17].

**Response for a Class**

Response for a Class (RFC) is similar to CBO but this measure also takes inheritance into count. It is the number of methods that can be executed as a response to a mes-sage received by class objects [13,17]. The count includes methods in the same class, methods accessible within the class hierarchy, and methods accessible in other classes. Source code with a high RFC count is complex and can be very difficult to trace potential code paths for testing and comprehension.

---

### Table 1: Metric Suites for Measuring Internal Quality Attributes

<table>
<thead>
<tr>
<th>Metric Suite</th>
<th>Description</th>
<th>Measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK Metrics</td>
<td>CK metrics designed specifically for object-oriented software [9]. Commonly utilized to predict fault-proneness [9] and included in static code analysis tools to provide automation opportunities.</td>
<td>• Weighted Methods Per Class (WMC)&lt;br&gt;• Depth of Inheritance Tree (DIT)&lt;br&gt;• Number of Immediate Subclasses (NOC)&lt;br&gt;• Coupling between Objects Classes (CBO)&lt;br&gt;• Response for a Class (RFC)&lt;br&gt;• Lack of Cohesion in Method (LCOM)</td>
</tr>
<tr>
<td>Robert C. Martin Metric Suite</td>
<td>This suite of metrics focuses on interdependence between packages, or cohesion [10]. It is also designed for object-oriented software.</td>
<td>• Afferent Coupling&lt;br&gt;• Efferent Coupling&lt;br&gt;• Instability&lt;br&gt;• Abstractness&lt;br&gt;• Normalized Distance from Main Sequence</td>
</tr>
<tr>
<td>McCabe’s Metric Suite</td>
<td>McCabe is most commonly associated with the concept of cyclomatic complexity. The metric was introduced as a way to quantify design decisions to indicate how difficult it is to test and maintain the method [11, 12]. Also commonly found in static analysis tools.</td>
<td>• McCabe’s Complexity&lt;br&gt;• Method Lines of Code&lt;br&gt;• Total Lines of Code&lt;br&gt;• Nested Block Depth</td>
</tr>
</tbody>
</table>

---

**Lack of Cohesion in Method**

The final metric in the CK suite is Lack of Cohesion in Method (LCOM). Since cohesion is considered a positive character-istic in object-oriented code, this metric considers the lack of cohesion. “A highly cohesive module should be independent” [13] of other modules and improve reusability. Lack of cohesion on the other hand signifies poor design that generates more complex code that is difficult to maintain. It is an indica-tor of potential redesign opportunities to create smaller more cohesive classes [13,17]. LCOM “has a significant effect on the total number of defects… and software development companies should concentrate on [it] to control… design defects” [8].

**External Metrics**

While internal metrics provide a wealth of interesting information, the measures themselves are not very helpful on their own. “For these early indicators to be meaningful, they must be related (in a statistically significant and stable way) to the field quality/reliability of the product” [7]. The field quality attributes can only be measured while the product is in the testing or operations and maintenance phase of the software lifecycle (ISO/IEC, 2002) and are known as external metrics. Figure 1 displays the relation-
HIGH MATURITY ORGANIZATIONAL CHARACTERISTICS

While much emphasis is placed on the development phase, most software products do eventually reach the operations and maintenance phase where they must then be maintained until the system is no longer needed or is replaced by a new product. It stands to reason, then, that software should be developed with maintenance in mind. But this is not typically the case because the software development team is focused on quickly creating a product to meet the needs of the customer at a cost and schedule the customer is willing to accept. The team is likely not considering how the product will be maintained after it is in operation.

Changeability

Changeability metrics relate to the actual maintenance activities that modify the existing code. The modification could be the result of discovering a defect, the need to adapt to a new environment, or the request for a new enhancement. Regardless of the reason for the change, most of the same qualities that make a code easier to analyze also makes it easier to change. High cohesion and low coupling are critical when developing an application with a focus on maintenance. An organization should review the relationship between internal metrics and changeability by gathering some of the following external metrics: change cycle efficiency, change implementation elapse time, modification complexity, parameterized modifiability, and software change control capability [19].

Stability

Most users dislike unexpected software behavior. Stability related metrics such as change success ratio and modification impact localization [19] provide evidence of unexpected behavior or the lack thereof. One method for decreasing the risk of unexpected behavior is to complete a thorough impact analysis and ensure modified code paths are systematically tested. With this understanding it is clear that the internal metrics that impact analyzability and testability are also crucial to predicting software stability.

Testability

The final maintainability component discussed is testability. As its name suggests, testability refers to the ease at which the software can be tested, or verified and validated. To measure testability, an organization should collect the following measurements: availability of built-in test function, re-test efficiency, and test restartability [19]. Coupling is a critical characteristic when determining testability [21]. When modules are coupled, it is more difficult or impossible to isolate the module under test to ensure the test is focusing only on the desired code and producing clear concise results. CBO and RFC metrics will have a strong correlation to the external metrics related to testability.

Conclusion

Software operations and maintenance is a critical phase in the software lifecycle. While much emphasis is placed on the development phase, most software products do eventually reach the operations and maintenance phase where they must then be maintained until the system is no longer needed or is replaced by a new product. It stands to reason, then, that software should be developed with maintenance in mind. But this is not typically the case because the software development team is focused on quickly creating a product to meet the needs of the customer at a cost and schedule the customer is willing to accept. The team is likely not considering how the product will be maintained after it is in operation.

However, when the same organization supports both software development and software operations and maintenance, there is incentive to keep maintenance in mind during the development process. It is also easier to capture both internal metrics, such as CK Metrics, that are available during the development phases as well as external metrics which are available only after the product is in operation. Internal metrics can help predict the level of support required to maintain the product once it is in operation. Furthermore, by analyzing the relationships between these two sets of metrics, the organization can improve both development and maintenance processes and thus improves the overall quality of the product as it supports the software from the cradle to the grave.