**Measuring the Complexity of OO Systems**

by [**Samudra Gupta**](mailto:guptasamudra@yahoo.co.in)

**What Is Complexity?**

Suppose you've got a particular implementation class that's become huge in size or too complex to maintain. Or, you've got a single class acting as the control class for a whole business layer, but there's too much business logic embedded within that one class. Or suppose again that you've been handed a class containing too much code duplication. These situations are what is referred to as "complexity." Learning how to use these three metrics may help steer you towards the correct re-factoring steps:

* Cyclomatic Complexity
* Response for Class
* Weighted methods per class

**Cyclomatic Complexity**

This non-OO metric, introduced by [**Thomas McCabe**](http://en.wikipedia.org/wiki/Cyclomatic_complexity), measures the structural complexity of a method. For easy understanding, take a look at this definition:

Cyclomatic Complexity (CC) = number of decision points +1

Decision points are conditional statements such as *if/else, while* etc. Consider the following example:

public void isValidSearchCriteria(SearchCriteria s){

if(s!=null) {

return true;

}else{

return false;

}

}

In the above example, *CC=2+1=3 (one if +one else +1)*.

Cyclomatic complexity has enormous impact on the testability and maintainability of code. As the example shows, if you want to test the *isValidSearchCriteria()* method, you have to write three tests to verify it: one for the method itself and two for the decision points within the method. Clearly, if the CC value increases, and there is an increasing number of decision points within any method, it means more effort to test the method.

The following table summarizes the impact of CC values in terms of testing and maintenance of the code:

|  |  |
| --- | --- |
| **CC Value** | **Risk** |
| 1-10 | Low risk program |
| 11-20 | Moderate risk |
| 21-50 | High risk |
| >50 | Most complex and highly unstable method |

From the complexity perspective of a program, decision points—such as *if-else*, etc.—are not the only factor to consider. Logical operations such as *AND, OR*, etc. also impact the complexity of the program. Consider the following example:

public void findApplications(String id, String name){

if(id!=null && name!=null) {

//do something

}else{

//do something

}

}

Using this example, you will need to perform three. Two for the normal *if-else* decision points and one for the logical *AND* operation.

Thus, in another version of cyclomatic complexity:

CC = no of decision points + no of logical operations +1

Following this version of cyclomatic complexity, the CC for the above sample code will be counted as:

CC = 2+1+1 = 4

Note that the number of tests you have to write for a specific method is equal to the CC measure of the method. Thus, a high CC value means more testing effort, which then means that the program itself is more complex to maintain.