

ALGORISMI

OO Legacy Code and Unit Testing

Miguel Lopez

June 2010

OO Legacy Code and Unit Testing

Legacy Code Issue

Michael Feathers' [Working Effectively with Legacy Code](#) introduced a definition of legacy code as code without tests, which reflects the perspective of legacy code being difficult to work with in part due to a lack of automated regression tests.

So, the question is: which unit tests to write first when I receive a legacy code? Do I have to write unit tests for each method? How can I identify the methods to test? In this post, we will answer those relevant questions by proposing an easy risk management approach.



Figure 1 - Legacy Code

Let's rephrase the problem. The issue is to identify the error-prone parts of the code without knowing this code. Once these poor quality pieces of code have been found out, we can focus our unit testing efforts. But, how can we locate the likely to fail parts of the code?

Simple Metrics Approach

To do so, we base our approach on two simple metrics of the methods:

- Cyclomatic complexity is the amount of statements like if, for, switch... in a method, and indicates the complexity of a method. The higher the cyclomatic complexity is, the more complex is the method, and the more error-prone is the method (See the left graph Figure 2).
- Coupling is the degree to which each function relies on each one of the other functions. The more dependent (coupled) is the function, the higher risk to retrieve bugs from other functions (See the right graph Figure 2).

ALGORISMI

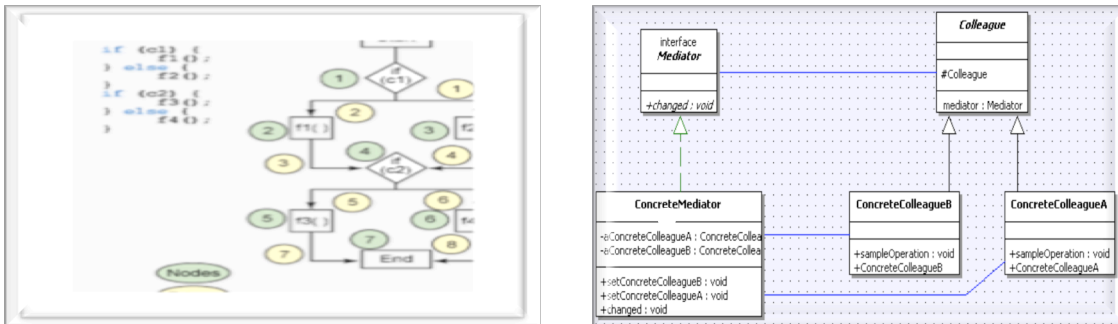


Figure 2 - Cyclomatic Complexity & Coupling

It is very difficult to obtain jointly good values for both metrics. Coding a simple (low cyclomatic complexity) and loosely coupled (low amount of calls to other methods) method is not that easy.

Even by doing this, it is not enough to reduce both metrics. Coupling will be increased by extracting a method. So, the best way is to reduce the cyclomatic complexity and the coupling at the same time. As such goal is a tedious task, most often code does not satisfy this rule. Complex methods are often loosely coupled, or strongly coupled methods are simple. But, the worst case is complex and strongly methods. This last type of methods must be at least (strongly) unit tested, or ideally reworked.

Anyway, sometimes, complex and strongly coupled methods are necessary. Business is such complex that there is no way to simplify the methods and to reduce coupling at the same time.

So, the only way to ensure quality is the unit testing technique. Regardless the difficulty to unit-test strongly coupled methods (mock objects,...), using both metrics (cyclomatic complexity and coupling) to identify the methods that must be strongly unit-tested is an easy way to prioritize the unit testing effort.

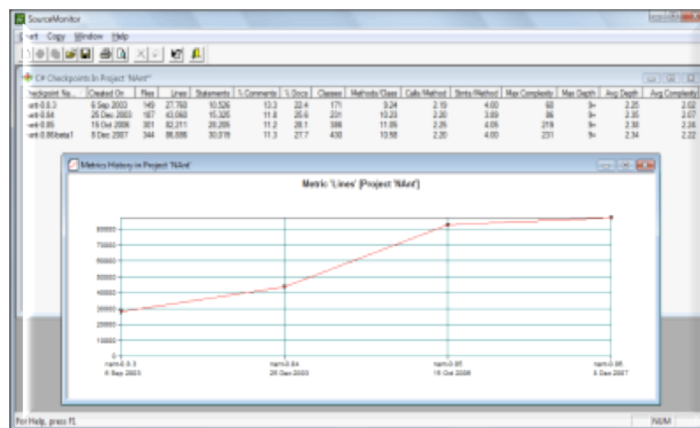


Figure 3 - SourceMonitor Results

So, analyzing the code with a freeware like SourceMonitor allows applying this method and in less than 1 minute, you will obtain a prioritized list of methods in regard with the unit tests. Just sorting the data table on the calls and complexity metrics, and you will have this list.