Andrei Fangli, class 248
March 30, 2015

Software Quality - Code Metrics

Assignment

Choose a Code metrics tool and apply it for your app. Your task is to produce a report about the use of the chosen tool. Results for your app, advantages, disadvantages.

Visual Studio for code metrics

Visual Studio comes with a code metrics tool. It contains five metrics: Maintainability Index, Cyclomatic Complexity, Depth of Inheritance, Class Coupling and Lines of Code. While metrics are just numbers and functions applied to them it can offer general guidelines and raise signals for places in code that are suspicious and need investigation. Next a brief description all code metrics will be presented to know what the numbers represent and how to interpret them.

Lines of Code

Probably one of the simplest metric can be used to indicate when methods or classes are "too large". Typically "too large" is a vague expression because it lacks precision. One can say that methods should be between 4 and 10 lines of code but there are always deviations from guidelines hence they are not rules. These deviations show up on results when methods go beyond a "reasonable" limit set by the team. Being able to navigate to the method and directly evaluate its complexity helps because some methods cannot be broken into smaller pieces because they do not make sense.

The same metric can work for classes. A good guide line is that classes should have single responsibilities, they should have one reason to change. Large classes tend to have more than one responsibility. Counting the number of lines per class can indicate violations of this principle.

Class Coupling

High coupling is usually a bad idea because the more coupled classes are the less likely it is for them to be reusable. If one object depends upon many then in order to reuse that one object there is need to know about all other depending objects.

This metric counts the number of classes used as parameters, return types and fields for all classes. The greater the variety the higher the coupling.

Depth of Inheritance

Class inheritance is one path towards code reuse, however inheritance most times breaks encapsulations, subclasses need hints about how the base class is implemented in order to make overriding effective. Inheritance is a great tool for reuse however it comes with some dangerous traps. Any change in a base class will affect all subclasses (flying rubber ducks anyone? – Head First: Design Patterns), this is a maintenance nightmare to add something in a base class then go through every subclass and override when necessary the new behavior. Not to mention to document this when new subclasses are written. The greater the class inheritance chain the harder it is to maintain. It is recommended to have small inheritance chains but it is not a rule, when large inheritance chains make sense then they should be used. The metric will signal it and a quick view of the classes involved should reveal whether the inheritance chain is showing its worth the potential maintenance issues it may bring.

Cyclomatic Complexity

This is a measurement of structural complexity. The total number of paths that code can take. More paths arise as if, switch and try catch statements are used. The optimum number for this metric is 1 showing no branches however this is hardly the case for real world application. Higher numbers are expected as applications grow however maintaining a low cyclomatic complexity at the method level is key to keeping the code easy to maintain and understand. When a method has a high cyclomatic complexity consider breaking it into smaller pieces thus lowering the complexity for each method.

The number is not relevant for classes, namespaces or projects as they are actually a sum of all measurements made for each method. The most relevant place to look at when evaluating this metric is at the method level.

Maintainability Index

This is a value computed through other metrics in an effort to determine a number that will indicate how hard it would be to maintain the code. The lesser the value is the harder it is to maintain the code. The higher, the better. The formula used for this metric is:

$$\mathrm{MAX}\begin{cases} 0 \\ (171 - 5.2 * \ln(HalsteadVolume) - 0.23 * (CyclomaticComplexity) - 16.2 * \ln(LinesOfCode)) * \frac{100}{171} \end{cases}$$