Using Code Quality Metrics in Management of Outsourced Development and Maintenance

Table of Contents

1. Introduction ... 3
1.1 Target Audience ... 3
1.2 Prerequisites ... 3
1.3 Classification of Sub-Contractors ... 3
1.3.1 Introduction ... 3
1.3.2 Individual Contractors ... 4
1.3.3 Groups of Contractors ... 4
1.3.4 Software Contractor ... 4
2. Managing the Project ... 5
2.1 Managing the Requirements ... 5
2.2 Managing the Deliverables ... 6
2.3 Service Level Agreements ... 6
3. The Metrics Approach ... 6
3.1 Metrics and McCabe IQ ... 7
3.2 Collection of Metrics ... 7
3.3 Visualization of Metrics and Source Code ... 8
3.4 Metrics in Traditional Development ... 8
3.5 Metrics in Agile Development ... 9
3.6 Metrics in the Maintenance Phase ... 9
3.6.1 In-House Maintenance ... 9
3.6.2 Outsourced Maintenance ... 10
4. Metrics Used ... 10
4.1 McCabe Metrics ... 11
4.1.1 Cyclomatic Complexity ... 11
4.1.2 Essential Complexity ... 11
4.1.3 Integration Complexity with derivatives S0 and S1 ... 11
4.1.4 Cyclomatic Density ... 12
4.1.5 Pathological Complexity ... 12
4.1.6 Branch Coverage ... 12
4.1.7 Path or Basis Path Coverage ... 12
4.1.8 Boolean or MC/DC Coverage ... 12
4.1.9 Combining McCabe Metrics ... 12
4.2 OO Metrics ... 13
4.2.1 Avg Cyclomatic Complexity for Class ... 13
4.2.2 Avg Essential Complexity for Class ... 13
4.2.3 Number of Parents ... 13
4.2.4 Response for Class (RFC) ... 13
4.2.5 Weighted Methods for Class (WMC) ... 13
4.2.6 Coupling Between Objects (CBO) ... 13
4.2.7 Class Hierarchy Level ... 14
4.2.8 Number of Methods in Class ... 14
4.2.9 Lack of Cohesion Between Methods (LOCM) ... 14
4.2.10 Combining OO Metrics ... 14
4.3 Grammar Metrics ... 15
4.3.1 Line Count ... 15
4.3.2 Nesting Levels ... 15
4.3.3 Counts of Decision Types ... 16
4.3.4 Maximum Number of Predicates ... 16
4.3.5 Comment Lines ... 16
5. Conclusion ... 16
6. References ... 17

McCabe Software, Inc.

(800) 638-6316 ● URL- http://www.mccabe.com ● 41 Sharpe Drive ● Cranston, RI 02920

1. Introduction

A metrics approach to facilitate the management of sub-contracted or outsourced development and maintenance has been successfully used by many organizations. For example, the U.S. Healthcare Finance Administration uses metrics to manage multiple contractors making and testing enhancements to regional versions of Medicare/Medicaid administration software, with a focus on the effectiveness of testing efforts. And various U.S. Department of Defense projects have used the approach to manage contractors developing weapons systems, with a focus on the quality of newly developed software.

The overall purpose of this document is to provide answers to several basic questions relating to the management of sub-contracted or outsourced development and maintenance:

- What is the metrics approach to managing sub-contractors?
- What is the quality of the code that has been developed by the sub-contractor?
- How well tested was the code before delivery by the sub-contractor?
- How are the ongoing maintenance costs related to the code quality?
- What is the role of automated tools?
- What is the role of visualization?

The word “sub-contracted” can be replaced with “outsourced” wherever applicable throughout the document.

1.1 Target Audience

This document is intended for anyone who has responsibility for, or intends to commence, a development or maintenance project that makes extensive or sole use of sub-contractors.

1.2 Prerequisites

- An understanding of the software development process
- Understanding of Service Level Agreements
- Understanding of the principles behind the CMM-I model (2)

1.3 Classification of Sub-Contractors

1.3.1 Introduction

This paper will discuss sub-contractors as belonging to one of three distinct groups:

- Individual Contractors working on-site
- Groups of Contractors working on-site
- Software Contractor working from their own premises both local and remote

McCabe Software, Inc.
(800) 638-6316 ● URL- http://www.mccabe.com ● 41 Sharpe Drive ● Cranston, RI 02920

Each of the above groups has to be viewed differently and thus the approach to managing their deliverables has to be approached from a slightly different metrics perspective.

In each case, the basic assumption being made is that at the end of a given project, the source code will be part of the deliverable. In the last group, the use of metrics in managing the end product even where the source code is not part of the deliverable will also be examined.

1.3.2 Individual Contractors

This group is perhaps the easiest to deal with. This group commonly has certain basic characteristics:

- Works on single-units of work individually or as part of an in-house team
- Managed using standard Project Management principles
- Control is by inspection
- Handover is usually informal, using code reviews and perhaps coverage reports

This group requires no special changes either to management techniques or the metrics retained and used during the product lifecycle.

1.3.3 Groups of Contractors

This will be a formal grouping of individuals from one software supplier assembled to work on a given portion or whole of a project. The prime contractor will employ the supplied individuals. This group commonly has certain basic characteristics:

- Work on mix of individual units and components
- Managed by using a combination of Standard Project/Man Management principles and Contract Management when working solely on a single larger set of deliverables
- Control is by inspection and ongoing metrics feedback
- Handover may be formal or informal depending upon the nature of the deliverable and group size
- There are strict acceptance criteria for larger deliverables
- This group will require extra effort in terms of the management load, and there should be greater focus on the metrics used to monitor their work. This will be especially true of metrics such as defect count.

1.3.4 Software Contractor

This is the more classic sub-contractor development model and increasingly in this decade, there has been a move to have the bulk of the work performed out-of-country in addition to being simply off the client premises.

This group has certain basic characteristics:

- Fixed Price contract
- Supplier has existing quality program
- Work on complete components and sub-systems
- Managed by Contract Management

- Control is by periodic on-site inspection, frequent deliverables and on-going metrics feedback
- Handover will be formal
- There are strict acceptance criteria defined in the contract
- Maintenance contract with same organization following delivery with a Service Level Agreement in place

The management of this group will require a conceptual/physical change to the management structure. A separate contract monitoring function will be required to interact between the users and the supplier, both to pass requirements and to act as the ‘referee’ during all phases of the project and specifically during the testing and acceptance phases. This group will make extensive use of metrics to monitor the course of the project and to monitor the deliverables, even when they do not include the source code.

2. Managing the Project

2.1 Managing the Requirements

One of the most important aspects to understand in managing a sub-contracted development team is the need for accurate defined requirements. Far too often when a project fails, the post-mortem shows that at the core of the problem was an inadequate set of requirements, which were subject to uncontrolled change during the life of the project. We must remember that an intrinsic part of software development is what can be termed as ‘programmer discretion’. This occurs when a programmer, reading an inadequate specification makes an arbitrary decision as to what might be required.

If you add to this mix the effect of cultural shift, which can sometimes occur when developing offshore, then inadequate requirement specifications will magnify any potential problems.

The requirements for the project/system/component must have been clearly worked through and documented in a mutually agreeable format and have been accepted as such by both parties. In addition, we need to consider the following:

- Questions must be documented and resolved by both parties.
- Change must be documented and agreed by both parties.
- The impact of change must be documented and agreed by both parties.
- The test plan produced must clearly reference the original requirements to ensure that there are no gaps in either the Test Plan or indeed the original requirements.
- Any test acceptance criteria must be updated to reflect any change to the original or subsequent set of requirements.
- Any test acceptance criteria must include code coverage to ensure that all the code supplied has been fully tested. This is especially important to ensure that not only does the supplied code perform as specified, but also there are no ‘hidden features’ within the code (including potential security holes), which have not been tested.

McCabe Software, Inc.
(800) 638-6316 ● URL- http://www.mccabe.com ● 41 Sharpe Drive ● Cranston, RI 02920

2.2 Managing the Deliverables

The contract sections that cover deliverables need to be far more sophisticated than simply specifying the delivery of a working component/sub-system. These sections need to make extensive reference to quality metrics to cover all possible types of component/system being delivered.

Metrics are used to determine a wide range of characteristics in the delivered software, covering time spent, defects uncovered during testing, defects fixed during testing, time to fix defects, code size, average complexity, code/branch/path coverage achieved during testing, code grammar, OO metrics that provide an indication of reuse, or possible need for refactoring, etc.

For each type of software deliverable, the appropriate metrics should be selected from the list provided in Section 4.

In order that the sub-contractor can comply with the terms of the metrics requirements in the contract, it is prudent that both parties make use of the same metrics gathering and reporting tool. This may necessitate the sub-contractor having his own licensed copy, which will be run within his software infrastructure. This will enable the sub-contractor to separate his development process from the managing organization, and thus achieve the required degree of technical separation and security.

2.3 Service Level Agreements

One of the modern methods for managing the maintenance aspect of the contract is using a Service Level Agreement (SLA). This is a contractual mechanism between a provider of services and a customer that defines a level of performance (13). Such an agreement defines in an empirical and measurable manner, the service to be performed, the level of service that is acceptable, the means to determine if the service has been provided at the agreed upon levels and any incentives for meeting agreed upon service levels.

One needs to understand that there is a difference between a Service Level Agreement and a traditional contract requirement. SLA's will contain more detail, describe incentives for meeting thresholds and specify incentives for meeting performance and quality of service requirements. Requirements are typically only measured at the end of a milestone or project. When non-compliance arises, typically the only recourse is to cancel the contract, terminate support and start again, which in the case of a large project is extremely difficult and expensive in terms of lost work. An SLA on the other hand contractually mandates the performance and quality attributes that all parties consider essential for the system to support the underlying business issue. SLA's codify explicitly the many factors that all the parties may implicitly assume, they will by specifying the quality metrics used enable all parties to have a clear idea of the expectations of the client. Measuring and monitoring performance and quality of service and product make it possible for an organization to judge whether the performance and quality requirements have been met, and in addition, such measurements support early detection and resolution of software quality issues. (7)

3. The Metrics Approach

In order to understand what metrics to collect and monitor from the three different types of sub-contractors, we will first review what metrics are available within McCabe IQ and the implications in using the various combinations.

McCabe Software, Inc.
(800) 638-6316 ● URL- http://www.mccabe.com ● 41 Sharpe Drive ● Cranston, RI 02920

3.1 Metrics and McCabe IQ

The range of metrics available within McCabe IQ covers some 146 different counts and measures, which can be further extended by the use of derived metrics and import of custom metrics from third party tools.

The metrics can be grouped according to six main 'collections' each of which provides a differing level of granularity and information about the code being analyzed. The collections are:

- McCabe Metrics Based on Cyclomatic Complexity (9)
- Execution Coverage metrics based on any of Branch, Path or Boolean coverage (9,11)
- Code Grammar metrics based around Line counts and Code Structure counts such as Nesting
- OO Metrics based on the work of Chidamber and Kemerer (3,4)
- Derived Metrics providing such abstract concepts as Understandability, Maintainability, Comprehension and Testability (6)
- Custom Metrics imported from third party software/systems, e.g., defect count

The specific metrics within each of these categories are further defined in section 4 of this document.

3.2 Collection of Metrics

McCabe IQ comes into play once source code has been produced. The code can be scanned prior to test as part of a code review process. Once the reviews are complete and any changes or refactoring has taken place, the test cycle can be commenced. Once the desired level of coverage is reached, the code is ready for integration testing and subsequent build and release. At that point, a snapshot of the metrics and the coverage achieved can be taken and sent to the Project Management Team for analysis using the management reporting functionality within McCabe IQ.

If code subsequently has to be reworked due to logic or performance shortcomings, then these snapshots provide the Project Management Team an understanding of how the code is being developed, and if the development processes stated in the contract are being followed.

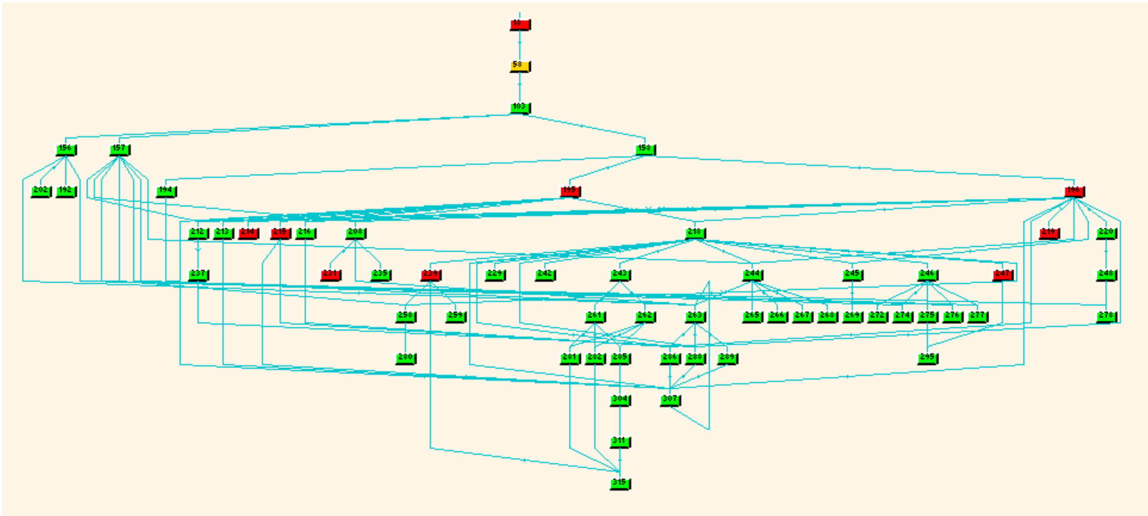
The metrics provided by the sub-contractor at each delivery, not only give an insight into the probable project end, but also provide an insight into the probable maintenance profile that may be developing in the project. This last factor is often ignored or simply not analyzed, such that when the delivered product moves into a conventional maintenance phase, the costs associated with that maintenance are excessive.

McCabe Software, Inc.

(800) 638-6316 ● URL- http://www.mccabe.com ● 41 Sharpe Drive ● Cranston, RI 02920

3.3 Visualization of Metrics and Source Code

Figure 1: Application Architecture View



At the load unit level, be it DLL, Library, EXE, etc., it is valuable not only to examine simple lists of metrics, or indeed comparisons of metrics between executable elements, but also to visualize the calling structure of the code, and graphical representations of the metrics generated by that code. Tools such as McCabe IQ can provide an insight into the architecture of the code, including the distribution of the complexity and any other of the many metrics that IQ gathers. Such a tool enables the Project Manager to understand the scale of any quality problem, whether it is localized to any particular element in the code, or perhaps more widespread throughout an application's architecture. McCabe IQ can be driven either from the command line, windows Icon, or directly from a range of the modern development IDE's such as Microsoft Visual Studio and Eclipse; and provides a wide range of views and reports, including aggregation of metrics across multiple load units and applications up to the enterprise. While one might undoubtedly argue that visualizing 10,000 methods is hardly going to improve understanding, by using McCabe IQ's drill down and exclusion features, one can easily get down to a low number of methods where problems may exist. This enables the user to then focus his attention on what matters and not lose focus in the morass of data that exists.

3.4 Metrics in Traditional Development

During the development process, metrics are typically collected relating to time and effort for any given phase of a project. As the project progresses, these metrics are measured against budgeted costs and the result is used to give an indication of how the project is performing against budget. These metrics when added to functionality completed and delivered gives the project office a degree of knowledge regarding the progress of the project against the original plan.

As part of the development, a quality plan will be implemented which will monitor the quality of the code as it is produced. It is in this area that code metrics will be used to ensure the code produced is not simply fit for purpose on the release date, but is constructed in such a manner as to facilitate change over the coming years.

McCabe IQ is run on completed components prior to inspection to ensure that the code conforms to industry standards for code quality. Metrics such as Cyclomatic Complexity, Essential Complexity, Nesting Depth, Counts of IF, Switch and Loops, Essential Density, Coupling, WMC and many more, are monitored to produce a baseline for the code release.

During the Testing phase, when following a test plan we monitor not just Requirements Coverage, which forms the basis of the Test Plan, but also Code Coverage, to ensure that we have tested all, or at least near all, of the executable code as part of our test plan. The Branch and Path coverage (including MC/DC for DO-178B (11)) Metrics are monitored and retained with the release as evidence of the effectiveness of the overall Test Plan.

3.5 Metrics in Agile Development

A key aspect of the Agile methodology is that the process changes to a progressive delivery of functionality delivered by continuous design, code, and testing through the development lifecycle. The initial design and estimates will not have been overly detailed as that would in part defeat the very nature of the Agile approach.

Here metrics are used to estimate with increasing accuracy our progress both functionally and more important perhaps, financially, through the development lifecycle. As each deliverable is completed and more detail of the complete picture is known, so the metrics gathered, both code and timescale can be used to build an accurate model of the project and how closely it is meeting its targets.

We must remember that as systems grow, whether expressed in terms of Function Points or other metrics, they start to exhibit increasing instability in the development phase:

- A greater percentage of the requirements will change between the project inception and project final delivery dates
- A greater chance of the project being cancelled
- Up to 50% of the functionality will either be rarely or never used

3.6 Metrics in the Maintenance Phase

Over the years, studies have shown that the major cost of a system lies not in its initial development but in the subsequent maintenance of that system during its life span.

One could also add the observation at this point that there are major differences between government and private sector projects. Government projects are often designed to last at least two decades if not three, while private sector projects are often designed to last less than a decade. Almost invariably, the larger private sector systems produced remain in use at least two decades and some considerably longer, particularly if you consider projects relating to Life Assurance.

In both scenarios, the need for metrics when using sub-contractors increases in importance with the size and maturity of the system. There are two main issues regarding metrics in the maintenance phase, dependent on whether the maintenance will be in-house or outsourced:

3.6.1 In-House Maintenance

In this scenario, metrics are most commonly used to determine the degree of difficulty of the change to be applied. The more complex the code and the less structured the code, the more difficult and costly the resultant change.

McCabe Software, Inc.
(800) 638-6316 ● URL- http://www.mccabe.com ● 41 Sharpe Drive ● Cranston, RI 02920

Here the metrics are visible, and thus if the original code was poorly written in terms of structure and complexity, there will be an inherent penalty during the maintenance phase.

3.6.2 Outsourced Maintenance

In this scenario, the code metrics may not be visible to the customer, and the cost of the change may well include the inevitable overhead for poorly designed and structured code. The customer, unless he has access to the code metrics has no real idea of what he is paying for, other than the changes seem rather more expensive than was originally expected with outsourcing. The hidden cost to the customer may well be the consequences of an inadequately built system, not merely the complexity of the change.

4. Metrics Used

McCabe IQ provides in excess of 100 individual metrics at the Method, Procedure, Function, Control, and Section/Paragraph level. In addition, there are a further 40 metrics at the Class/File and Program level. Further metrics can be derived using the tool to enable site-specific requirements to be generated. The subsections which follow discuss the following categories of metrics:

- McCabe metrics
- OO metrics
- Grammar metrics

The threshold values described in the individual metrics have been taken both from the references papers (3, 4, 9, 10, 11) and from a wide range of practical experience and study. Note these threshold values are suggestions; that is, the precise threshold values selected by you are not as critical as the idea that you choose certain quality metrics to measure, and that certain review actions are taken when your thresholds are exceeded.

To select which metrics and thresholds to use very much depends upon your specific requirements. The following will introduce a selection and explain what the metrics can be used for, in both development and maintenance phases. Typically code and quality metrics are not used individually but in combination, and where this is the case mention will be made at the end of the relevant section.

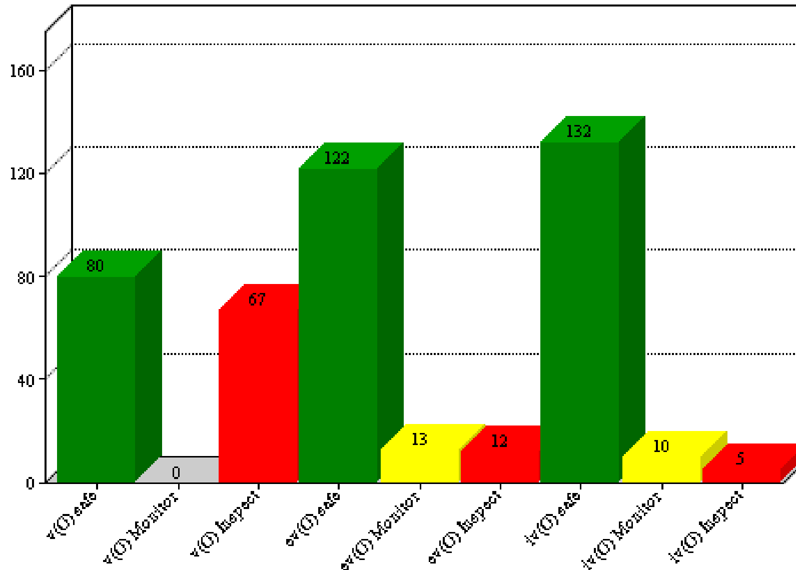
One must always remember that when collecting metrics you are relying upon subordinates who need to ‘buy into’ the metrics program, therefore it is important to only collect what you intend to use. Do NOT use secret metrics and above all do not use metrics in any covert measurement of individuals keeping in mind ‘The Hawthorne Effect’ that when you collect metrics on people, the people being measured will change their behavior. Either of these practices will destroy the efficacy of any metrics program.

McCabe Software, Inc.

(800) 638-6316 ● URL- http://www.mccabe.com ● 41 Sharpe Drive ● Cranston, RI 02920

4.1 McCabe Metrics

Figure 2: McCabe Metrics Quality Gates



4.1.1 Cyclomatic Complexity

A measure of the amount of logic in a code module, in 3rd & 4th generation languages this can best be defined as a Method, Procedure, Function, Control, or Section/Paragraph depending upon the programming language. If Cyclomatic complexity is excessively high, it leads to impenetrable code, which is higher risk due to difficulty in comprehension and testing. The commonly used threshold is 10. When the Cyclomatic complexity of a given module exceeds 10, the likelihood of the code being unreliable is much higher. A high Cyclomatic complexity indicates decreased quality in the code resulting in higher defects that become costly to fix.

4.1.2 Essential Complexity

A measure of the degree to which a code module contains unstructured constructs. If the Essential complexity is excessively high, it leads to impenetrable code, which is higher risk due to difficulty in comprehension and testing. Furthermore, the higher value will lead to increased cost due to the need to refactor, or worse, reengineer the code. The commonly used threshold is 4. When the essential complexity of a given module exceeds 4, the likelihood of the code being unmaintainable is much higher. A high essential complexity indicates increased maintenance costs with decreased code quality. (In place of Essential Complexity, some organizations have used the Essential Density metric, which is a ratio of the Essential to the Cyclomatic complexity.)

4.1.3 Integration Complexity with derivatives S0 and S1

A measure of the interaction between modules of code within a program. S0 provides an overall measure of the size and complexity of a program's design, without reflecting the internal calculations of each module. S0 is the sum of all the integration complexity in a program ($\sum iv(g)$). S1 is (S0 –

Number of methods + 1). This is primarily used to determine the number of tests for the 'smoke test' that is designed to ensure the application would execute without issues in module interaction.

4.1.4 Cyclomatic Density

A measure of the amount of logic in the code expressed as a ratio of decisions to the lines of executable code. By eliminating the size factor, this metric reduces complexity strictly to modules that have unusually dense decision logic. The higher the Cyclomatic density value, the denser the logic. The metrics should be in the range of .14 to .42 for the code to be simple and comprehensible.

4.1.5 Pathological Complexity

A value greater than one indicates poor coding practices, such as branching INTO a loop or INTO a decision. It must be noted that these conditions are not easy to replicate with modern post 3GL languages. Pathological complexity represents extremely unstructured code, which indicates poor design or construction and a high recommendation of reengineering the code.

4.1.6 Branch Coverage

A measure of how many of the branches, or decisions in a module, have been executed during testing. McCabe IQ by default uses Expanded Branch Coverage, which extends the traditional measure by ensuring the logical conditions within a decision statement are tested, for those languages where the compiler implements short-circuiting. (Optionally, McCabe IQ can ignore the short-circuiting when analyzing Branch Coverage.) If the Branch Coverage is <95% for new code or 75% for code under maintenance, then the test scripts require review and enhancement.

4.1.7 Path or Basis Path Coverage

A measure of how many of the basis (Cyclomatic) paths in a module have been executed. Path Coverage is the fundamental of McCabe design and indicates how much logic in the code is covered or not covered. This technique requires more thorough testing than Branch Coverage, and should be considered for components with high quality requirements. If the Path Coverage is <90% for new code or 70% for code under maintenance, then the test scripts require review and enhancement.

4.1.8 Boolean or MC/DC Coverage

A technique used to establish that each condition within a decision is shown by execution to independently and correctly affect the outcome of the decision. This technique is more often used in safety critical systems and projects that require to be mandated by the FAA under DO-178B (11). When used, the requirement is for 100% Coverage, which in part explains the very high cost of such systems.

4.1.9 Combining McCabe Metrics

While Cyclomatic Complexity is the basic indicator for determining the complexity of logic in a unit of code, it is typically used in conjunction with other metrics for determining certain other key characteristics (10). Following are suggestions:

Code Review Candidate

If the Cyclomatic Complexity exceeds 10 and Essential Complexity exceeds 4 (or Essential Density exceeds .4), then the unit needs review.

Code Refactoring
If the Cyclomatic Complexity exceeds 10 and the algorithm (v(g) – ev(g) < ¼ v(g)) is true, then the code is a candidate for refactoring.

Inadequate Comment Content
If the plot of Cyclomatic Complexity against Comment % (in terms of Lines of Code) does not show a linear increase, then the outliers require their comment content to be reviewed.

Test Coverage
If the plot of Cyclomatic Complexity against Path Coverage does not show a linear increase, then the outliers require their Test Scripts to be reviewed.

4.2 OO Metrics

4.2.1 Avg Cyclomatic Complexity for Class

If the average is greater than 10, then this measure indicates a high level of logic in the methods of the class, which in turn indicates a possible dilution of the original object model. If the average is high, then the class should be reviewed for possible Refactoring.

4.2.2 Avg Essential Complexity for Class

If the average is greater than one, this may indicate a dilution of the original object model. If the average is high, then the class should be reviewed for possible Refactoring.

4.2.3 Number of Parents

If the number of parents for a class is greater than one, this indicates a potentially overly complex inheritance tree.

4.2.4 Response for Class (RFC)

The RFC is the count of all methods implemented within a class plus the number of methods accessible to an object of this class due to implementation. The larger the number of methods that can be invoked in response to a message, the greater the difficulty in comprehension and testing of the class. Low values indicate greater specialization. If the RFC is high, making changes to this class will be increasingly difficult due to the extended impact to other classes/methods.

4.2.5 Weighted Methods for Class (WMC)

WMC is the count of methods implemented in a class. It is recommended WMC does not exceed the value 14. This metric is used to predict the effort required to re-write or modify the class. The aim is to keep this metric low.

4.2.6 Coupling Between Objects (CBO)

Indicates the number of non-inherited classes this class depends on, and thus the degree to which this class can be re-used. This measure tends to be high in DLL's, where the software is deployed as a complete entity, but should be low for executables, where re-use is to be encouraged. Strong coupling increases the difficulty in comprehending and testing a class. The aim is to keep this less than 6.

4.2.7 Class Hierarchy Level

Indicates degree of inheritance used, if greater than 6, the increased depth increases the testing effort and if less than 2, the value indicates a poor exploitation of OO. One should aim for a high percentage of 2 & 3 across the application.

4.2.8 Number of Methods in Class

If greater than 40, the value indicates that the class has too much functionality and might be split into several smaller classes. Ideally, one should aim for no more than 20 methods in a class.

4.2.9 Lack of Cohesion Between Methods (LOCM)

This metric measures the dissimilarity of methods on a class by instance variable or attribute. The percentages of methods in the class using an attribute are averaged and subtracted from 100. The measure is expressed as a percentage, if the percentage is low, this implies simplicity and high re-usability; if the measure is high, indicates class is a candidate for refactoring and could be split into two or more subclasses with low cohesion.

4.2.10 Combining OO Metrics

While Cyclomatic Complexity is the basic indicator for determining the complexity of logic in a unit of code, it can also be used in the evaluation of OO systems. Here it is used in conjunction with OO metrics to determine suitable candidates for Refactoring (11):

Refactoring

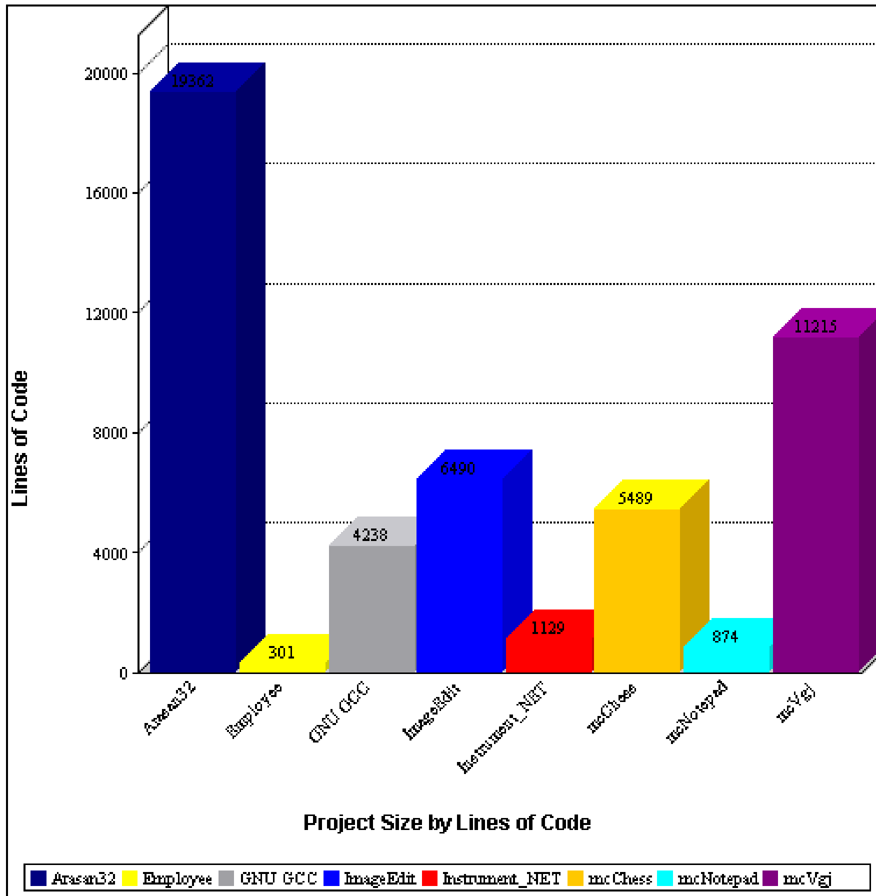
- If the Average Cyclomatic Complexity is high (> 10) and the number of Methods is very Low (<10) then, the class requires refactoring.
- If the Average Cyclomatic Complexity is low and the Lack of Cohesion is high, then the class is a suitable candidate for refactoring into two or more classes.
- If the Average Cyclomatic Complexity is high and the Coupling Between Objects is high, the class is a candidate for refactoring.
- If the Coupling Between Objects is high and the Lack of Cohesion is high, the class is a candidate for refactoring.

McCabe Software, Inc.

(800) 638-6316 ● URL- http://www.mccabe.com ● 41 Sharpe Drive ● Cranston, RI 02920

4.3 Grammar Metrics

Figure 3: Component Size Comparison



4.3.1 Line Count

An indicator of size. It is however commonly used in estimation algorithms such as COCOMO 2(1), and in measuring defects as a ratio of number of defects/1000 lines of code.

4.3.2 Nesting Levels

Indicates the level of intricacy and algorithm in a unit of code. Nesting Levels can indicate unnecessarily complex conditions, which may make future modification increasingly difficult.

When used with IF, Switch and Loop constructs, this metric can indicate unnecessarily complex code that might be suitable for Refactoring. With certain algorithms, the measure indicates multi-dimensional array processing. The industry standards are typically 4, 2 & 2 for the constructs listed.

4.3.3 Counts of Decision Types

Used to enumerate Single outcome (IF & loop) and Multiple outcome decision statements. When used together with Cyclomatic Complexity, the value can determine if a Method/Procedure/Control/Section is overly complex and a suitable candidate for Refactoring.

4.3.4 Maximum Number of Predicates

This measure indicates overly complex decision statements, which are candidates for Refactoring.

4.3.5 Comment Lines

Indicates the level of comments in a unit of code. This measure is usually used as a component to such metrics as understandability (6), or the 3MI & 4MI. The measure can also be used in the ratios of comment lines to lines of code or comment lines to Cyclomatic Complexity to give an indication documentation levels within the code. In common with all code analyzers, McCabe IQ cannot validate the accuracy of the comments; this can only be established by detailed or random inspection. Historically it has been accepted that a ratio of between 15-25% of comments is considered adequate to enable a third party to understand what the code is attempting to perform.

5. Conclusion

This paper has provided a document on how to use certain metrics to manage the process of using sub-contractors on a project. In specifically defining the different types of sub-contractors, it is hoped that minds are now focused more clearly on how to approach what are three very differing management scenarios. In addition, the use of code metrics provides a clearer insight into what you the customer, might really be paying for when the code is in, or moves downstream in the product life-cycle to the maintenance phase.

The main conclusions to be drawn from this paper are:

- Without proper code metrics, you, the manager, have at best an incomplete picture of what is being produced.
- Without metrics automation using a tool such as McCabe IQ, you, the manager, face a complex process in focusing on the potentially problematic areas of your code base.
- Without metrics based code visualization using a tool such a McCabe IQ, you, the manager, face a complex process in identifying and reviewing the code in the problematic areas of your code base.
- Without a metrics-reporting tool such as McCabe IQ, you, the manager, face a complex task in managing the information that any analysis of the code base will provide.
- Without the use of industry-accepted metrics and their associated thresholds for code, you, the manager, face an uphill task in setting the expectations of your own management chain.

This paper provides guidelines for the all the above issues in a simple and concise manner and can thus answer the question of how McCabe IQ could be used to help manage sub-contractors.

McCabe Software, Inc.
(800) 638-6316 ● URL- http://www.mccabe.com ● 41 Sharpe Drive ● Cranston, RI 02920

6. References

Boehm B., Clark B., Horowitz E., Madachy R., Shelby R., Westland C., “The COCOMO 2 Software Cost Estimation Model”, International Society of Parametric Analysts, May 1995.

Software Engineering Institute, Carnegie-Mellon University, CMMIS for Systems Engineering and Software Engineering (CMMI-SE/SW, V1.1).

Chidamber, S.R. and Kemerer, C.F., “Towards a Metrics Suite for Object Oriented Design”, Proc. of the 6th ACM Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA), 1991, Phoenix, AZ, pp. 197-211.

Chidamber, S.R. and Kemerer, C.F., "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol. 20, pp. 476-493, 1994.

Curtis W., “Management and Experimentation in Software Engineering”, Proc. IEEE Vol. 68 No 9, September 1980.

Dept of Army, Software Test and Evaluation Guidelines, pamphlet 73-7, 7/25/1997.

ISO/IEC 9126: Information technology - Software Product Evaluation - Quality characteristics and guidelines for their use - 1991.

Gaines T., and Michael J B., Service Level Agreements as Vehicles for Managing Software Acquisition of Software-Intensive Systems, Defense Acquisition Review Journal.

Halstead M., “Elements of Software Science”, North Holland 1977.

McCabe T.J., “A Software Complexity Measure”, IEEE Trans Software Engineering Vol2, December 1976.

Rosenberg L, “Applying and Interpreting OO Metrics”, NASA, SATC, April 1998.

RTCA SC- 167 / EUROCAE WG-RTCA, Inc. “Software Considerations in Airborne Systems and Equipment Certification”, RTCA/DO-178B, December 1992.

Sturm R., Morris W. & Jander M. (2000). Foundations of Service Level Management. Indianapolis, IN: Sams.

McCabe Software, Inc.

(800) 638-6316 ● URL- http://www.mccabe.com ● 41 Sharpe Drive ● Cranston, RI 02920