White Paper

# McCabe Recommended Approach to Code Reviews

Table of Contents

"Why do you see the speck in your neighbor's eye, but do not notice the log in your own eye?"

The quote above forms the principle behind code reviews. The benefits obtained from performing code reviews are many and will be explained in this paper, but it is important to understand the basic concept.

This simply states that a group of people reviewing code will find up to 82% of the errors within the code prior to the commencement of testing (IBM). In addition 80% of the lifetime cost of a piece of software goes into maintenance, and hardly any software is maintained by the original author (Sun Microsystems 1995-1999).

**Who Should Read This Paper**

This document is intended for anyone who has responsibility for, or intends to commence, a program of Code Reviews within their organization. It is primarily aimed at those in a supervisory position who will be responsible for the quality of a system or group of systems.

## Purpose of this Paper

This document has been written to provide the answer to three basic questions:

- What is the function of code reviews in increasing productivity and code quality?
- What is the McCabe approach to code reviews?
- How can McCabe IQ be used to set up an automated code review process?

## Code Review in the Software Industry

### Purpose of Code Reviews

Organizations that implement Code Reviews do so to achieve two distinct yet at the same time overlapping objectives, which are:

- To achieve a reduction in errors during the development process, in order to reduce wasteful testing time and subsequent (and costly) production errors (Fagan 1979 & 1986,Boehm 1981 & Kaplan 1995)
- To reduce future maintenance costs by ensuring a standardized software solution

These two objectives form the basis of all code reviews and can be further expanded as follows:

- Regular code reviews are a powerful tool in the development process, as developers tend to overlook mistakes that they have created themselves.
- Code reviews are sometimes used to ensure that the code produced conforms to a given 'standard' in terms of its basic grammar, constructs and complexity, thus helping to simplify the future maintenance of this code.
- Code reviews help to spread expert knowledge throughout a development team. The suggestions and ideas which evolve during review sessions often enable the group to take big steps forward in terms of the technology and techniques used.
- Code reviews are often used to ensure that a deliverable to a client meets the standards defined in the contract. Conversely, code acquired from a third party to be maintained in-house is often subject to a code review to ensure that it meets the standards defined in the contract.
- In more recent times the concept of refactoring, in languages such as Java, has further enhanced the code review process. The review process can now be used to make the reviewed code easier to understand and this broader comprehension leads to even more useful ideas. When taking these ideas as a source for an immediate refactoring to be implemented by the reviewers, code reviews can deliver concrete results.

## Common Approaches to Code Reviews

There are a number of approaches used for code reviews. They can be categorized into three groups: *peer reviews, automated code checking for rule based compliance,* and *automated metrics based* reviews.

### Peer Reviews

This is the oldest and most common form of code review used in the industry at present. Peer reviews range from simple line-by-line reviews to structured walkthroughs to the latest Fagan inspections and refactoring reviews. These techniques have a number of common challenges:

- Manual process
- Time consuming
- Objectivity
- Clash of Egos
- Require very careful planning if they are to be successful.

**XP**

The most extreme form of the peer review process forms an integral part of what is termed Extreme Programming (abbreviated "XP"), where two developers craft a piece of code together, one in the coder role, the other in the reviewer role, and thus the review process is in real time with the code production phase.

Ideally this process involves the author of the code sitting down in a room with one or more reviewers and examining some or all of the code they have produced. All parties will work according to a clearly defined plan of action and identify any errors in the logic or deviation from installation standards. These are noted down in an action plan and at the start of the next code review a check is made to ensure that all the remedial action has been applied.

The only divergence from the above pattern is with refactoring, where the changes are applied and tested as part of the review process.

## Automated Code Checking

In this approach a tool is used to perform a syntax check on the code and identify any deviation from a predetermined set of coding standards. The coding standards will be dependant upon the language used, but a selection of what the reviewer will be checking for includes:

- Code layout
- Usage of Comments
- Conformance to label and variable naming conventions i.e. Hungarian
- Identification of Control constructs
- Conformance to internationally accepted programming standards i.e. MISRA
- Mixed Mode arithmetic
- Initialization of variables
- Indexes, Pointers and subscripts correctly initialized before usage and tested correctly
- Identification of Dead Code
- Identification of dangerous coding practices.

This form of review is currently the most popular form of automated review for which there is a wide range of products available. The main drawbacks of this approach are threefold:

- Excessive number of errors in legacy code
- Different tool required for each language
- Different results formats and reports for each language
- Selection or modification of programming standards.

**The Example of "Cyclomatic Complexity"**

Cyclomatic complexity is a measure of the number of logical paths (decision sequences) within a function of code. Independent studies indicate that defects exponentially increase for functions that have more than 10 logical paths. Of all metrics in the industry, cyclomatic complexity has the highest correlation to defects, thus it is an important measurement to track.

**Automated Metrics Based Reviews**

In this approach a tool is used to perform a check on a wide range of metrics that measure not just the grammar of the code but also its basic characteristics in code engineering terms. Metrics are either collected (by measuring such elements as lines of code, the number of logical paths, and the number of 'children') or calculated (derived from measurements using mathematical formulas). Using the resulting values, reviewers examine such factors as Cyclomatic Complexity and Unstructure (McCabe 1976 & 1979), Software Science (Halstead 1977) and Composite measures of Software Complexity (Curtis 1980).

Metrics are generally collected and calculated on a module-by-module basis and used to highlight areas of code suitable for further manual inspection. Metrics can be made indicators of code quality by establishing meaningful thresholds for them. Numerous studies as well as standards published by the National Institute of Standards and Technology (NIST) have provided guidelines for key metrics thresholds (some will be discussed later in this paper). However, each organization generally creates a suitable set of thresholds appropriate to its development environment and code base.

This paper will focus on the following subset of metrics:

| | |
|---|---|
| • Cyclomatic Complexity | (McCabe) |
| • Essential Complexity | (McCabe) |
| • Integration Complexity | (McCabe) |
| • Program Size | (Halstead) |
| • Comment Density | (General) |
| • Logic Density | (McCabe) |
| • Nesting Depth, including Switch Depth & Loop Depth | (General) |
| • Maximum Number of Predicates in a single statement | (General) |
| • Fan-in and Fan-out ratios | (McCabe) |

The range of metrics that can be measured is very large, but most organizations typically use some or all of the above.

## The Challenges of Code Reviews

There are a number of distinct challenges when using or attempting to implement a code review process, particularly in terms of productivity, objectivity, and approach.

## Productivity

Code Reviews are seen as unproductive, typically doubling the time it takes to 'craft' the code in the first instance. It has however been identified by a wide range of studies, both by individuals (Boehm, Kaplan, Gilb) and corporations such as IBM (Santa Rosa & Rochester), that the cost of the removal of errors in the coding phase can be as low as 1/92 of the cost of removal in the Customer Release phase.

The other issue with Productivity is the sheer volume of code that may need to be reviewed. In the ideal world, all the source code would be inspected (as the requirements of CMM level 3 and up dictate). In practice, however, this is generally not feasible and some way has to be found to enable the reviewer to focus only on the code that needs to be examined as a matter of urgency, for example, because it is at a higher risk for developing defects in future revisions or because it is particularly costly to maintain. This would leave the bulk of the code either to be examined on a random basis, if this is legacy code and resources are insufficient, or at least prepare a prioritized sequence for the full code inspections that are required as we move up the CMM scale.

**Using International Standards**

The selection of metrics and thresholds in the interests of setting objective standards can be a very contentious process. It is for this reason, as well as to simplify the process, that organizations increasingly turn to internationally recognized coding standards, though sometimes with minor variations to reflect their own environment.

Some internationally recognized coding standards include: C (MISRA), Java (Sun Microsystems), VB (Microsoft Programming Conventions for VB), Cobol (IBM Programming Standards for Cobol).

Using such standards has the added benefit of easing the movement of development staff in the industry, the use of sub- contractors or even outsourcing, as more and more organizations turn to internationally agreed standards.

## Objectivity

This is perhaps the greatest single issue in Code Reviews, namely the difficulty in persuading developers that what they have crafted may need to be altered to conform to a common norm. Developers have two basic issues in regards to Code reviews at a personal level.

- Someone else wrote the code so why should they be criticized for the way it was written
- They wrote the code and how dare someone criticize their craft. Coding is an art, not a science

These are admittedly extreme positions, however they provide a somewhat frank start to the issue of objectivity. All too often code reviews get side tracked into personal issues, especially where a 'loose' code review process is being followed.

The challenge here is to adopt a common set of standards so that all agree on what is required and how the process should be run - specifically, to agree on a set of coding standards and metrics thresholds. Both need to be chosen carefully to fully reflect the environment in which the organization is working.

The aim is for inspections to approach what has been described as "ego-less programming" (IBM).

## Approach

The process by which a code review or inspection is carried out will to a great degree determine the degree of success and acceptability of the whole concept of Code Reviews within an organization. The various papers on this subject recommend a team of up to four people known as inspectors, each performing a distinct task with one member of this group as the author. The aim is to identify defects and log them, with no attempt to correct them 'on the fly'.

- The inspectors describe each defect in about 7 words or less using simple English
- The inspectors do not determine how to fix the defect, this is the responsibility of the author.
- There should be no discussion as to whether the defect exists; once it is logged, it is a defect.
- The author is not allowed to explain, describe or defend their work, except in response to a direct question.
- The inspectors must be trained in the task.

The correction process is entirely in the hands of the author, who at a pre-determined time must show the corrections to the chief moderator.

**A Common Interface**

In addition to addressing the challenges of approach, productivity, and objectivity, the McCabe IQ approach addresses one of the major limitations of automated code reviews - that they are language specific. By contrast, McCabe IQ provides a common interface for most major languages, including C, C++, Cobol, Fortran, Java/JSP, M204, Perl, PL1, and VB. This is particularly important as the number of languages used in any given development environment is continually growing.

## The McCabe IQ Approach to Code Reviews

McCabe IQ tool provides up to 105 different code metrics and is therefore ideally suited to implement a metrics-focused code inspection regime.

The McCabe IQ approach is to identify the methods, functions, controls or perform ranges in the code that exceed the selected thresholds for any of the metrics that have been selected by the organization. This enables the inspection team to focus their attention on the code that requires further examination. It also enables the developers to review their code as part of the development process to ensure that the code will meet the requirements of the organization before the code is inspected.

The aim of this closed loop process is to increase productivity and reduce the subjective element from the code review process, and thus address a number of the major challenges of Code Reviews.

The McCabe IQ approach to code reviews consists of three interlocking elements:

- Selection of metrics
- Selection of thresholds
- Identification of “outliers” - code that exceeds the lower and upper thresholds.

## Selection of Metrics

It is important that the organization initially selects a small set of metrics that can be agreed upon by both QA personnel and developers. This is done to fine tune the process and get buy-in from all parties. Once the process is accepted and up and running, further metrics can be added as required.

The initial selection should include:

| | |
|---|---|
| • Cyclomatic Complexity | (McCabe) |
| • Essential Complexity | (McCabe) |
| • Integration Complexity | (McCabe) |
| • Comment Density | (General) |
| • LOC | (McCabe & Halstead) |

Some of the additional metrics that can be added later include:

| | |
|---|---|
| • Program Volume | (Halstead) |
| • Nesting Depth, including Switch Depth & Loop Depth | (General) |
| • Maximum Number of Predicates in a single statement | (General) |
| • Number of Unique calls excluding Library routines | (McCabe) |
| • Logic Density | (McCabe) |
| • Fan-in and Fan-out ratios | (McCabe) |

**The Purpose of Derived Metrics**

Derived metrics are designed to aid supervisors who may not necessarily wish to have the level of detail provided by the individual metrics. Derived metrics are also designed to address code quality in general, abstract terms in order to make the Code Review concept more acceptable to those whose background is not, or interest may not be, in Software Engineering.

In addition, McCabe IQ can calculate derived metrics, based on a combination of metrics available in the product. In order to simplify the code review process McCabe provides four derived metrics indicative of:

- Size
- Unreadability
- Unstructure
- Modular Size.

Each of these is calculated from three or more of the previously defined metrics, and each is subject to a modification factor. Their use must be carefully calibrated in order that the modification factor will truly reflect the characteristics of the code.

## Selection of the Acceptable Thresholds for Each of the Metrics

**More on Thresholds**

The full range of thresholds for individual metrics is covered in the separate McCabe paper titled "Metrics and Thresholds in McCabe IQ."

The selection of thresholds is key to the code review process, especially when dealing with legacy code. Thresholds allow reviewers to identify where the code quality has degraded to unacceptable levels. The following table details industry standard thresholds for the metrics mentioned earlier in this paper.

| Metrics | Lower Threshold | Upper Threshold |
|---|---|---|
| Cyclomatic Complexity | 10 | 15 |
| Essential Complexity | 4 | 8 |
| Integration Complexity | 4 | 8 |
| Comment Density | <20% | <15% |
| Logic Density | .14 | .42 |
| Fan-in and Fan-out ratios | (Site Dependent) | |
| LOC | 20 | 50 |
| Program Size | (Site Dependent) | |
| Nesting Depth, including Switch Depth & Loop Depth | 4 | 6 |
| Maximum Number of Predicates in a single statement | 4 | 6 |
| Number of Unique calls excluding Library routines | (Site Dependent) | |

Typically, organizations have an initial threshold selection process, followed by periodic reviews of not just the code but also of the thresholds.

## Identification of the Code that Exceeds the Lower and Upper Thresholds ('Outliers')

The process to identify code that exceeds the lower and upper thresholds starts with the creation of a Custom Report within McCabe IQ that produce lists of methods, procedures, controls or Perform Ranges which exceed the limits given above.

The custom reports may take many forms, from simple lists with multiple metrics, to Kiviat diagrams with multiple metrics, to Scatterplot graphs where we plot the distribution of code against two metrics (see "McCabe Functionality for Code Reviews" later in this paper).

Developers can use the Custom Reports to monitor their own progress and ensure their code satisfies the code quality criteria for the organization. QA team/Code Inspectors can use the Custom Reports to ensures that their attention is focused on the 'questionable' code, leaving the bulk of the code for examination in the standard maintenance cycle.

## The Review Process

### Reporting

McCabe IQ is used to create reports that give a snapshot picture of the code. A typical selection of reports for the code review process would be comprised of:

- Scatterplot of Complexity vs. Structure Distribution
- Scatterplot of Complexity vs. Comment Density
- EQ Quality Report.

The reports above conveniently answer the key questions of Size, Understandability, Structure, and Modularity.

Once the reports have been produced, the review team can examine them in order to identify the code that requires manual inspection. The review team would then manually inspect the 'questionable' code, draw up a list of defects, and produce a schedule for the correction of defects. This Code Review would occur typically before the commencement of Unit testing and again when the code is handed over into production.

**Automated Reporting**

The use of an automated process to identify the code that requires manual inspection increases the productivity of the Code Inspection process, thus satisfying one of the challenges stated earlier. In addition this automation also removes any personal agendas from the Code Review process, thus satisfying the second of the major challenges in Code Reviews.

### Visualization

McCabe IQ, through its visualization capabilities, provides the review team with the capability to obtain a detailed visualization of any component that has been identified as requiring closer inspection. McCabe IQ displays each module of code, not only in its native programming language format, but also more importantly, with its logic diagram by its side.

The detailed inspections can be performed in one of three ways:

- Using IQ generated reports and the context sensitive module names
- Using the Printed reports and the Find functionality
- Using the Graph/ASL tool to step through all the components, stopping only at those whose logic graph appears over complex.

The last of the three techniques is best suited for reviews carried out either by the developer or by the direct supervisor, prior to a formal review.

## McCabe Functionality for Code Reviews

### McCabe Metrics

As has been stated earlier in this paper, McCabe IQ has the capability to record up to 105 metrics for a given set of source code. These metrics are gathered at the module (function/method/Perform Range) level, Program Level (one or more source files), or System Level (one or more programs). In addition there is the full range of OO metrics as defined by Chidamber and Kemerer (1991 & 1994).

**More on Metrics**

This paper does not address the full capabilities of the McCabe Metrics. For more information, read the McCabe paper titled, "Metrics and Thresholds in McCabe IQ."

### McCabe Reporting

The McCabe/IQ tool comes with three discrete reporting capabilities:

- Standard Reports provided with the tool, comprises some 35 reports, both textual and graphical
- Custom Reports that can be generated from the Standard reports using the in-built Report Generator to produce text reports, graphs, or files suitable for input to spreadsheet packages
- McCabe/EQ, which makes of an embedded RDBMS to create and manage a database of metrics for a given analysis or set of analysis. This is supplied with a set of standard reports and a powerful report generator. For further details on this option refer to the McCabe web site at www.mccabe.com.

### McCabe Visualization

McCabe IQ provides extensive visualization capabilities at all levels of an analysis. The basic McCabe IQ visualization tools are:

- Battlemap - a high level view of all components analyzed and their inter-relationships (calling hierarchy)
- Graph/ASL listing - the low level view of an individual component or module, showing side by side, the native programming language format, and the associated logic diagram
- Scatterplot - Distribution Graph plotting any two metrics, when used within the tool is context sensitive and can be used as the first stage in a search process
- Kiviat Diagrams - Distribution Graph plotting up to five metrics.

**Visualization Controls**
With the exception of the Graph/ASL listing which only works at the module level, all the other visualization tools can be at whatever level of selection the user requires. McCabe IQ also provides extensive grouping capabilities to enable a user to reduce the scope of the initial analysis according to a pre- determined criteria based on file name, functionality, variables, metrics, and so on. This then enables Code reviewers to focus their efforts accordingly.

## McCabe Automation

Most aspects of the code review process using McCabe IQ can be automated with the McCabe IQ Command Language capability (CLI).

The McCabe IQ CLI command is "CLI METRICS". It can be executed from the command line using the following parameters:

| | |
|---|---|
| -pcf | McCabe PCF name |
| -output | Output file to hold Results |
| -report | Name of Custom report to be run |

For example, the following runs a Custom Report called "quality_level1" against the Code analyzed in c:\CV6\mcChess.pcf, and it places the resulting report in c:\listings\qual_rep1.txt.

```
CLI METRICS –pcf c:\CV6\mcChess.pcf –output c:\listings\qual_rep1.txt –report quality_level1
```

## References

Boehm B. *Software Engineering Economics*. Prentice-Hall 1981.

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.

Fagan M.E. "Design & Code Inspections to Reduce Errors in Program Development", *IBM Systems Journal* Vol. 15, No. 3 1976.

Fagan M.E. "Advances in Software Inspections", *IEEE* 1986.

Halstead M. *Elements of Software Science*, North Holland 1977.

Kaplan, C., R. Clark, and V. Tang. 1994. *Secrets of software quality: 40 innovations from IBM. New York*, McGraw Hill.

McCabe T.J. "A software Complexity Measure" ,*IEEE Trans Software Engineering* Vol2 December 1976.

Sun Microsystems Inc. ,*Java Code Conventions*. 1995-1999.