How They Do It in Switzerland… Outsource the Code, Insource the Quality

McCabe IQ Analyzes and Visualizes Software Quality in Critical International Financial Systems

Introduction

This paper examines how a major Swiss financial services company analyzes and monitors the quality of its software developed with its offshore development partner.

The software being maintained forms a critical part of the Swiss financial infrastructure and is maintained with the now legendary quality for which Switzerland is renowned the world over.

The implemented process is based around the measurement and monitoring of 20+ quality metrics for COBOL and 50+ quality metrics for Java, covering both conventional software metrics and code grammar observations. These metrics are based on industry standards with the aim of providing both partners with a software quality benchmark to work towards. The process is, apart from the source code download from the CM repository, entirely automated up to the point of report production.

Initial Analysis

Following an external review, the Swiss software company commissioned a firm of consultants to review their Software Development Life Cycle (SDLC) and suggested a series of quality indicators that should be used to monitor the quality of the code being maintained and produced. Due to both the volume of legacy code and new code being written in an ever changing financial landscape, this became an imperative if costs and rework were to be minimized over time.

The volumes of code are substantial, comprising some 24MLOC of COBOL (expanded COPY) and some 2.5MLOC of JAVA, both growing at a rate of 2% or more annually.

Formal naming standards were reinforced with equally formal coding standards to produce a sound base from which to start. The enforced naming standards helped greatly in automating the whole process, enabling analysis by component to be produced with limited human intervention.

There are two major releases each year, and each is analyzed and the metrics trends monitored.

The Quality Indicators

As stated earlier, there are 20+ COBOL and 50+ Java quality indicators, too many to list in a short paper, but to give a feel what is being measured, here are a few for each programming language:

COBOL

Q-Metric (1):	***There must be no undocumented COBOL programs.***
Q-Metric (2):	***There must be no undocumented COBOL copy books.***
Q-Metric (3):	***Comment to line count ratio of a COBOL program must not be below 15%.***
Q-Metric (4):	***Comment to line count ratio of a COBOL copy book must not be below 15%.***
Q-Metric (5):	***COBOL programs must not be longer than 2000 code lines.***
Q-Metric (6):	***COBOL copy books must not be longer than 1000 code lines.***
Q-Metric (7):	***COBOL paragraphs must not have a cyclomatic complexity greater than 15.***

Q-Metric (8):	***COBOL sections (without embedded paragraphs) must not have a Cyclomatic Complexity greater than 15.***
Q-Metric (9):	***Modules must not have too deep control flow nesting depth (depth > 4).***

Java

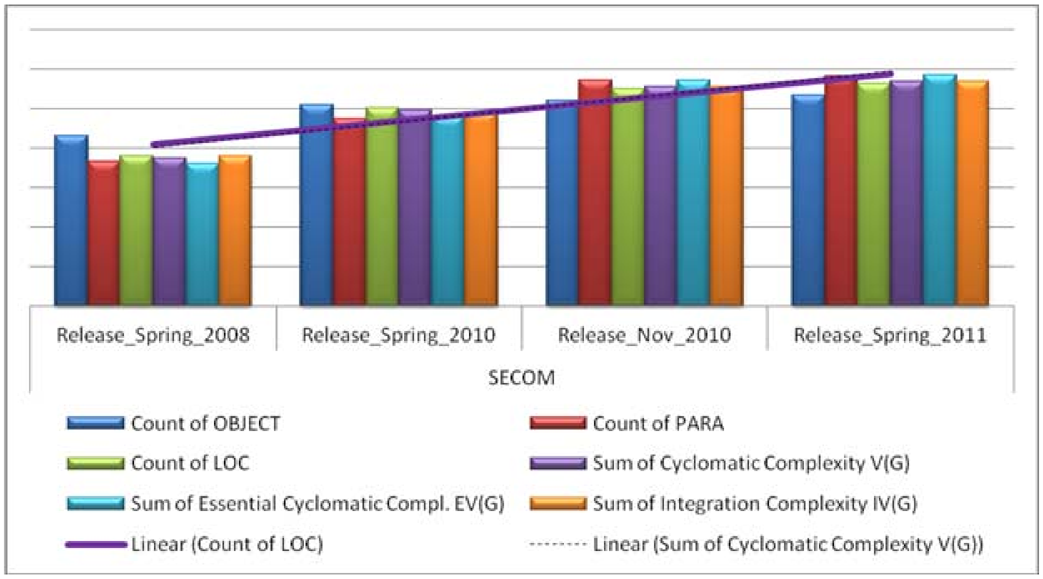
Q-Metric (1):	***The length of an artifact name must not be less than 2 characters.***
Q-Metric (2):	***The length of an artifact name must not be greater than 50 characters.***
Q-Metric (3):	***A code fragment of 40 lines or more must not occur identically somewhere else in the system.***
Q-Metric (4):	***Every package must contain at most 50 public classes or interfaces.***
Q-Metric (5):	***Source files must not be longer than 2000 LOC.***
Q-Metric (6):	***Cyclic dependencies between classes are forbidden.***
Q-Metric (7):	***A method should not contain more than 200 Lines of Code (LOC).***
Q-Metric (8):	***The cyclomatic complexity of a method should be less than 10.***
Q-Metric (9):	***Methods must not have too deep control flow nesting depth (depth > 4).***
Q-Metric (10):	***Class/Package names must match the regular expression ([a-z][a-zA-Z0-9]+)+***
Q-Metric (11):	***There must not be missing `default` branches in `switch` statements.***
Q-Metric (12):	***There must not be any empty catch blocks.***

Using the mix of source code analysis and Java grammar checking, and an XML interface, the metrics are combined in one database with a single GUI. This single database is then used to aggregate the metrics and derive the trending data, thus enabling teams to focus on the high risk, high incident components when they are identified.

The Software

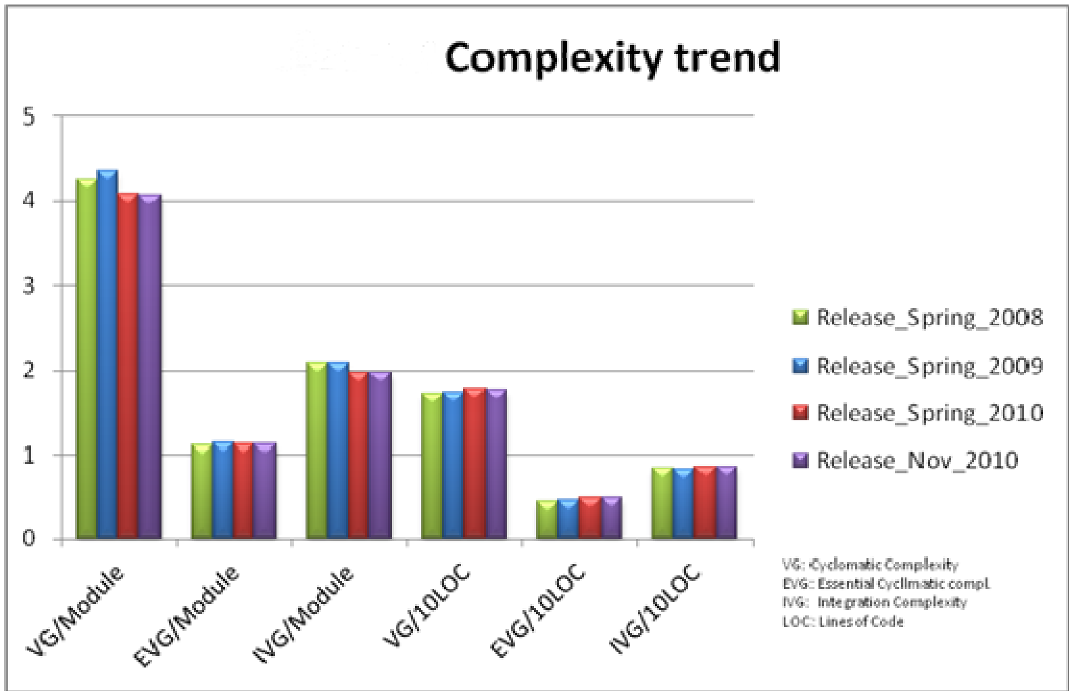
The Swiss financial services company selected McCabe IQ as the source code analysis tool of choice for both COBOL & Java, for its graphical insights into the code, ability to provide the widest range of metrics both inbuilt and derived, and ability to be fully scripted. In addition, McCabe IQ was shown to be flexible enough to allow extensions to be added for the more 'exotic' quality indicators and further programming languages to be added in the future if required. Checkstyle was chosen as the Java grammar checker of choice. McCabe IQ has been further extended to process the observation output from Checkstyle, convert the data to code metrics and import these metrics into both the McCabe IQ project and enterprise databases.

The tools are driven by a hierarchy of Windows Command files that unpack the source files into their component folders, build a McCabe IQ analysis, process any log files and upload the accumulated metrics into the McCabe IQ enterprise database. Further processing takes place, providing management an unambiguous view of trends by component, by system, and by release.



The Results

Reports can now be produced to focus on particular areas of concern or risk. The development partner has the information they require to improve code quality when that particular code is modified. The reports are both textual and graphic, summarizing the vast quantity of data into a series of pie and bar charts. Red or amber arrow indicators are added manually to provide a more human touch.



Summary

When needing to analyze and monitor the software quality of their substantial and growing code base, this Swiss financial services company selected McCabe IQ, based not only on its unrivalled metrics capability, extensive visualization and scripting for ease of use, but also on its inherent flexibility in accepting metrics from other processes. McCabe IQ provides the management team with a 'birds eye' view of their code quality irrespective of programming language and method of gathering the data. The implemented process is simple to run and requires little manual intervention, giving the Swiss owner and their offshore partner industry standard metrics to view within their joint process and helping each party understand what is happening with the code quality over time.

Item	Nov-Year 1	**Spring Year 2**	
COBOL Source programs	6,844	7'008	
Thousand Lines of Code	19,220	19,801	
COBOL Copybook programs	45,999	47,538	
Thousand Lines of Code	5,418	5,616	
Avg. Complexity/Paragraph	4.07	4.08	↗
Avg. Structuredness/Paragraph	1.13	1.13	➡
Avg. Design/Paragraph	1.97	1.97	➡