Cantata provides a full and unique suite of intelligent testing capabilities for the efficient unit and integration testing of C and C++. In this feature brief we highlight the ways in which the Static Metrics features of the tool can be best used.
Contents

Static Metrics..................................................................................................................................................1

1 Cantata Static Metric Analysis Overview..................................................................................................3

1.1 PURPOSE OF CANTATA STATIC METRIC ANALYSIS..........................................................................3

1.1.1 Code Complexity.................................................................................................................................3

1.1.2 Object Oriented Metrics.....................................................................................................................3

1.1.3 Coding Standards................................................................................................................................4

1.1.4 Test Effort Planning............................................................................................................................4

1.2 PERFORMING STATIC ANALYSIS WITH CANTATA........................................................................4

1.2.1 Detailed use of Cantata Static Analysis Metrics................................................................................4

1.3 THE METRICS AVAILABLE...................................................................................................................5

1.3.1 Standard Code Size Metrics.............................................................................................................5

1.3.2 Standard Code Quality Metrics.........................................................................................................5

1.3.3 Standard Complexity Metrics...........................................................................................................5

1.3.4 Specialist Object Oriented Metrics.....................................................................................................6

1.3.5 Additional System Metrics................................................................................................................7

1.4 GENERATING STATIC METRICS..........................................................................................................7

1.5 PROCESSING THE DATA.........................................................................................................................7

1.6 VISUALISING THE RESULTS..................................................................................................................9

Copyright Notice

Subject to any existing rights of other parties, QA Systems GmbH is the owner of the copyright of this document. No part of this document may be copied, reproduced, stored in a retrieval system, disclosed to a third party or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of QA Systems GmbH.

© QA Systems GmbH 2019
1 Cantata Static Metric Analysis Overview

Even software which has been thoroughly dynamically tested can still have its problems. Static analysis gives the developer useful information on non-functional qualities of the source code, such as its maintainability and compliance with coding standards. Static Analysis, as the name suggests, involves the static inspection of the source code in order to provide an assessment of various non-functional features relating to the software.

1.1 Purpose of Cantata Static Metric Analysis

Cantata static analysis provides over 300 metrics on source code which can assist developers in four important areas:

- Objective measurement of code complexity and structure
- Object Oriented metrics assessment
- Enforcement of coding standards
- Test effort planning

1.1.1 Code Complexity

Cantata supports code complexity metrics as a means of increasing the maintainability of software, through objective measurement using recognised ‘academic’ and common sense metrics:

- Halstead’s Software Science metrics.
- McCabe’s, Myers’ and Hansen’s cyclomatic complexity metrics.
- Average and maximum nesting level.
- Basic counts of language constructs (comments, lines of code, statements, parameters etc)

1.1.2 Object Oriented Metrics

In addition to code complexity measures, Cantata also provides a number of metrics which measure aspects of object oriented implementation. These include:

- Chidamber and Kemerer’s MOOSE metric set.
- Fernando Brito e Abreu’s MOOD metric set.
- Bansiya and Davis’ QMOOD metric set.
- Robert Martin’s object-oriented dependency metrics.
- McCabe’s object-oriented metrics.
- Bansiya’s class entropy metrics.

All metrics are provided at the function, class, translation unit, or system level, as appropriate.
1.1.3 Coding Standards
Organisations are increasingly adopting coding standards as a means of improving software quality and maintainability. However, unless these standards can be verified in an automated way, it is difficult to enforce them effectively. While Cantata is not a coding standards rule checking product, it does provide the developer with static analysis metrics on the use of several useful coding constructs such as:

- Unreachable code
- Switch statements with no defaults or fallthroughs
- Number of GOTO statements, used and unused GOTO Labels
- Lack of cohesion methods

1.1.4 Test Effort Planning
Static analysis is often performed as part of a code review exercise, prior to dynamic testing. One important feature of Cantata static analysis is the capability to use industry standard complexity metrics to accurately estimate the testing effort for source items. An example is McCabe Cyclomatic Complexity and its variants, which the result of which strongly correlates to the minimum number of test cases required to achieve 100% decision code coverage.

1.2 Performing Static Analysis with Cantata
Cantata static analysis is invoked on a build of a Cantata enabled software project.

By default Cantata will analyze all code in the project for all 300+ static analysis metrics. Configuring the analysis can be achieved through the following methods:

- Enabling / disabling analysis of system headers
- Specifying which static analysis metrics to calculate (through an options file)
- Specifying the source files, functions or classes to be analyzed
- Specifying precisely the source code statements to be analyzed (through pragmas in the source code)

1.2.1 Detailed use of Cantata Static Analysis Metrics
Cantata Static Analysis provides hundreds of function, class and file scope code quality and complexity metrics. Code quality and complexity metrics provided by Cantata can help users to determine areas of the code that will most likely suffer from bugs, as well as producing data from which the time required for testing can be estimated. Once the metrics have been gathered by Cantata they can be processed and manipulated using an add-in for Microsoft Excel.
1.3 The Metrics Available

Some of the more common metrics provided by Cantata are described below. For an exhaustive list please refer to the Cantata Static Analysis manual.

1.3.1 Standard Code Size Metrics

These are simple metrics regarding the number of lines of code, comments, etc.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINE_CODE</td>
<td>Total number of lines of code (including blank lines and comments).</td>
<td>Function or system</td>
</tr>
<tr>
<td>LINE_COMMENT</td>
<td>Total number of lines of comments (both C and C++).</td>
<td>Function or system</td>
</tr>
<tr>
<td>LINE_SOURCE</td>
<td>Total number of lines of source code (not including blank lines or comments).</td>
<td>Function or system</td>
</tr>
</tbody>
</table>

1.3.2 Standard Code Quality Metrics

The quality of a piece of software is to some degree based on the number of occurrences of dubious code contained within it. These metrics alert the user of such occurrences.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>LABEL_GOTOUSED</td>
<td>Number of goto labels that are used.</td>
<td>Function or system</td>
</tr>
<tr>
<td>LABEL_GOTOUNUSED</td>
<td>Number of unused goto labels.</td>
<td>Function or system</td>
</tr>
<tr>
<td>STMT_GOTO</td>
<td>Number of goto statements.</td>
<td>Function or system</td>
</tr>
<tr>
<td>SWITCH_NODEF</td>
<td>Number of switch statements with no default.</td>
<td>Function or system</td>
</tr>
<tr>
<td>SWITCH_FALLTHRU</td>
<td>Number of non-empty case blocks which fall through to the next case block.</td>
<td>Function or system</td>
</tr>
<tr>
<td>UNREACHABLE</td>
<td>Number of statically unreachable statements in the given scope.</td>
<td>Function or system</td>
</tr>
</tbody>
</table>

1.3.3 Standard Complexity Metrics

The complexity of a piece of code is generally regarded as a measure that will affect the effort involved with maintaining it. These metrics attempt to estimate the complexity of the software based on various factors, such as the level of nesting.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>HALSTEAD_PARAMS</td>
<td>Number of parameters.</td>
<td>Function</td>
</tr>
<tr>
<td>MCCABE</td>
<td>The McCabe Cyclomatic Complexity value for the function.</td>
<td>Function</td>
</tr>
<tr>
<td>NESTING_MAX</td>
<td>Maximum statement nesting level.</td>
<td>Function</td>
</tr>
<tr>
<td>NESTING_SUM</td>
<td>Sum of the statement nesting levels for all statements in the function.</td>
<td>Function</td>
</tr>
</tbody>
</table>
1.3.4 Specialist Object Oriented Metrics

Many standard metrics are still applicable to OO systems. For example, the maximum nesting levels within functions is also applicable to class methods. However there are also a range of specific OO metrics. These may be with respect to a given class, or for the system as a whole.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_DEPTH</td>
<td>Maximum length of inheritance path to ultimate base class.</td>
<td>System</td>
</tr>
<tr>
<td>MOOD_AD</td>
<td>Number of new attributes defined for this class.</td>
<td>Class</td>
</tr>
<tr>
<td>MOOD_MD</td>
<td>Number of new methods plus overridden methods defined for this class.</td>
<td>Class</td>
</tr>
<tr>
<td>MOOD_AHF</td>
<td>Proportion of attributes that are hidden (private or protected).</td>
<td>Class</td>
</tr>
<tr>
<td>MOOD_MHF</td>
<td>Proportion of methods that are hidden (private or protected).</td>
<td>Class</td>
</tr>
<tr>
<td>MOOSE_CBO</td>
<td>Level of coupling between objects. The number of classes with which this class is coupled (via a non-inheritance dependency from this class to that, or vice versa).</td>
<td>System</td>
</tr>
<tr>
<td>MOOSE_WMC_MCCABE</td>
<td>Average McCabe Cyclomatic Complexity value of for all methods of the class excluding inherited methods defined in this translation unit.</td>
<td>Class</td>
</tr>
<tr>
<td>MOOSE_LCOM98</td>
<td>Chidamber &amp; Kemerer’s Lack of Cohesion of Methods metric (1998 definition). The minimum number of disjoint clusters of (new or overridden) methods (excluding constructors), where each cluster operates on disjoint set of (new) instance variables.</td>
<td>Class</td>
</tr>
<tr>
<td>MOOSE_RFC</td>
<td>Chidamber &amp; Kemerer’s Response for a class metric. The number of methods or functions defined in the class or called by methods of the class.</td>
<td>Class</td>
</tr>
</tbody>
</table>

The ‘OO’ aspects of the C++ language have tended to render the old procedural C metrics less useful, but fortunately new sets of metrics have taken their place. The popular ones include MOOSE (Metrics for OO Software Engineering), MOOD (Metrics for OO Design), and QMOOD (Quality Metrics for OO Design). Between them they define a number of metrics which can be useful for judging whether a C++ class is ‘worth testing’. Some examples are:
1.3.5 Additional System Metrics
Additional system level metrics can be created by taking averages for various class or function scope metrics. For example, we can calculate the mean McCabe Cyclomatic Complexity value for all functions or methods within our system.

1.4 Generating Static Metrics
Cantata will generate the static analysis metrics for units as they are compiled. Gathering static analysis data does not actually require the code to be compiled but this is a side effect of Cantata being called during the compile phase.

1.5 Processing the Data
The data contained within the CSV files is raw and needs to be processed and combined to generate many of the higher-level metrics. The task of merging and combining the raw CSV data is controlled by the Cantata Static Analysis Results Presentation Tool, which is a Microsoft Excel Template spreadsheet.

If you are not running Cantata on Windows but have access to Excel on a Windows machine, the Static Analysis Results Presentation Tool can be installed separately by running the Windows installer. If you do not have access to Excel, the same functionality can be achieved using the csvmerge tool and the analyse AWK script from the bin directory.

Once loaded, you will be presented with the following screen:

<table>
<thead>
<tr>
<th>Quality identified in source code</th>
<th>EXAMPLE METRICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poor or Questionable Design</td>
<td>‘MCCABE Quality’</td>
</tr>
<tr>
<td></td>
<td>‘MOOSE Lack of Cohesion among Methods’</td>
</tr>
<tr>
<td></td>
<td>‘MOOD Attribute Hiding Factor’</td>
</tr>
<tr>
<td>Estimated Number of Faults</td>
<td>‘MOOD Methods Defined’</td>
</tr>
<tr>
<td></td>
<td>‘MOOD Attributes Defined’</td>
</tr>
<tr>
<td></td>
<td>‘MOOSE Weighted Methods in Class’</td>
</tr>
<tr>
<td>General Complexity</td>
<td>‘MOOSE Depth of Inheritance’</td>
</tr>
<tr>
<td></td>
<td>‘QMOOD Number of Ancestors’</td>
</tr>
<tr>
<td></td>
<td>‘MOOSE Number of Children’</td>
</tr>
<tr>
<td>Estimated Test Effort</td>
<td>‘Methods Available’</td>
</tr>
<tr>
<td></td>
<td>‘MOOSE Response for a Class’</td>
</tr>
<tr>
<td></td>
<td>‘MOOSE Coupling Between Objects’</td>
</tr>
<tr>
<td></td>
<td>‘MOOD Method Hiding Factor’</td>
</tr>
</tbody>
</table>
Click the “Click Here to Start” button to begin. You will then be asked to select a set of CSV files for processing. You will then have to confirm that your selection is complete. Click “Yes” to continue. The Summary screen should then be displayed. This contains a summary of all the processed, combined and formatted metrics gathered for each different scope:
Click on the other tabs to see the break down of results for that particular scope. For example, click the Function tab to see the individual values for each of the metrics with regard to each of the methods contained:

![Graph showing static metrics results]

The buttons at the base of each metric label can be used to filter the rows that are displayed.

### 1.6 Visualising the Results

Although the formatted textual static analysis results are very useful it is often more applicable to visualise the data within a graph. Plotting graphs of the data can aid understanding and create overall pictures of the trends occurring lower down that are not immediately obvious when you are reading the textual data.

Graphs are easy to create in Microsoft Excel but first we need to create a table of the data we would like to plot. Return to the Menu tab of the spreadsheet and click the “Visualisation” button. This will display a dialog offering scopes for the data that you can use. Generally these will include:

- Class
- Function
- Category

To begin with, select the “Function” option.

Now you must select the metrics to plot. You should select a set of procedural style metrics that will evaluate code size, quality and complexity for all of the methods/functions in the code:
Click “OK” to continue. You will now have the option to select the chart type. The “Standard Charts” are all of the chart types available within Excel, which is a considerable set. The “Kiviat Chart” is specific to the Cantata Template and resembles a radial graph:

Click “OK” to continue.

If you elected to use one of the standard charts then a table of the specific data will be produced:
Click the chart icon to start creating the graph:

Select the 3D Column graph and make sure that the data table is in an order where the bars will generally get higher toward the rear of the graph.

This will give you something that looks like this:

The diagrams below were produced, using the Cantata Static Analysis Presentation Tool.
Cantata Feature Brief: Static Metrics

Class Kiviat plot for 2 classes

MOOD_AD
Attributes Defined

MOOSE_DIT
Depth of Inheritance

MOOD_MHF96A
Method Hiding Factor

Complexity graph for 2 classes

stack() and bounded_stack() Class

MOOD_AD
MOOD_MHF96A
MOOSE_DIT
bounded_stack
bounded_stack::full_push
stack
stack::empty_pop
stack::new_error
stack::node

Lower Percentile Value
Middle Percentile Value
Upper Percentile Value

Code and Commenting Standards Graphic

convert ()
Function
range_ok ()

CPP_COMMENTS
C_COMMENTS
LINE_CODE
UNREACHABLE
SWITCH_NODEF
SWITCH_FALLTHRU
STMT_GOTO
LABEL_GOTOUNUSED
LABEL_GOTOUNUSED