Krakatau
Software Metrics Tool

14 June 1999

Prepared by:
Lockheed Martin Mission Systems
Mark K. Lutey

CONTENTS

1.0 INTRODUCTION

1.1 Overview

The RSC is an “integrated COTS toolkit” which results in a complex system. Complexity makes sustainment difficult and therefore software quality is a must. Quality software can be defined in many ways such as reliability, maintainability, stability, testability, etc. But, without some form of measure how can one define quality other than on an esoteric sense? To provide measurement for software quality this paper proposes the use of an automated tool called Krakatau by Power Software.

1.2 Executive Summary

As CERES and RSC are called upon to support more and more customers there will be a need for an increase in system reliability. Additionally, programs such as Mongoose will require CERES and RSC to evolve their systems. To meet increasing demand and evolution in the most economical way our software must be reliable, easy to maintain and test, and modular for reuse. To determine if the software meets these standards some form of measure must be made. One such approach that is recommended to make these measurements is using a software tool called Krakatau. There is no system risk using Krakatau, as it is a standalone product that runs off-line and analyses source code. The major advantage is the tool provides some 70 metrics that will provide insight into the form and function of our software.

2.0 REQUIREMENTS

2.1 Description of Requirements

A common approach to improving the run-time performance of a software system is to measure various run-time properties of the components of the system to find “bottlenecks” that account for the majority of the “time cost” of the system. By analogy, a promising approach to reduce life-cycle costs of software development has been to measure properties of software components to find “complexity” bottlenecks that account for the majority of the difficulty of developing or maintaining the software [LAMB1]. By correcting defects in the software at the definition and development phase it has been shown that there can be an order of magnitude in cost savings as to correcting defects in the maintenance phase [PRESS17]. In order to realize these cost savings it is recommended that measures be taken of software components while in the early stages of d evelopment.

Doing software measures by hand would be time consuming. Using automated tools would reduce the man-effort required. To develop such tools by one’s self would be time consuming and prone to error. By using a commercial automated tool one re alizes the benefits of having the tool immediately, no production cost, and that the commercial tool is subjected to much more scrutiny by others to ensure that the tool is correct. One such tool is Krakatau produced by Power Software.

Krakatau can only examine software written in C/C++ and Java. Such a tool cannot examine G2 and commercial software products such as Satellite Tool Kit. Despite that limitation the majority of our software problems lay in the middleware which can be examined by Krakatau.

As a side note, the examination of the rest of the system software will be in a real-time mode. For example, G2 provides software components that are currently available to obtain run-time performance metrics of the IMT software. Sun Microsystems provid es ProCtool and Symon that are compatible with the Solaris operating system and they can measure the run-time performance of system software. We are currently using ProCtool but it is not officially supported by Sun. Symon can measure the run-time performance of any machine from one location and is supported by Sun. We are currently evaluating Symon for possible use. To get the full functionally of Symon additional licenses are required. The Talarian RT software has software components that can examine the middleware run-time performance. G2, ProCtool, and Talarian software components are currently in the RSC baseline.

2.2 Need Event

In order to realize development cost savings benefits the Krakatau tool should be acquired prior to the next major software build. Once Krakatau is obtained we can have immediate insight into the existing middleware, mission unique and SYS500 software.

3.0 IMPLEMENTATION APPROACH

3.1 Design Overview

A comparison was done among several software metric tools as seen below in Table 1. Freeware metric tools are available, such as CCCC, but they are not guaranteed to work or to be fixed if a bug is found. Therefore freeware tools are not included.

Table 1. Software Metrics Tool Comparison

	Testworks/ Advisor	**McCabe QA**	**Krakatau**	**CodeCheck V8.0**	**QualGen**	**Resource Standard Metrics (RSM)**
Languages	Ada, C, C++, FORTRAN	Ada, C, C++, FORTRAN, Java, Model 204, Visual Basic	C, C++, Java	C, C++	Ada, C++	C, C++, Java
Platform	Solaris	Solaris	Solaris	UNIX	?	Solaris
Use w/Config Mgt tools	Yes	Yes	Yes	?	?	No
Structure charts & flowgraphs	No	Yes	Yes (Krakatau +)	No	No	No
Metrics	17	60+	70+	Few McCabe & Halstead	200+ (mostly for Ada syntax)	McCabe and LOC
Graphs	One	Yes	Yes	Print out histogram	Print out histogram	No
GUI	Yes	Yes	Yes	No, you write code with their tool	No, you write code with their tool	Command line

Metric tools CodeCheck and QualGen require that we write our own programs using their tools. As such, these two tools were immediately eliminated since this requires add additional labor on our part to type new programs and then debug them if errors exist. This development time will add to the cost of our software. Additionally, both of these programs focus primarily on checking syntax and not quality. There are other tools available to check syntax and adherence to coding standards such as “lint” for C. Finally, there is no graphical interface using these two tools. Therefore additional features such as “help”, on-screen displays and ease of use are not available.

RSM was eliminated since it has no graphical interface and few software engineering metrics.

Between Testworks/Advisor, McCabe QA, and Krakatau, only Krakatau provides a rational set of software metrics with a graphical users interface (GUI) at a reasonable cost.

The Krakatau tool is a standalone product and would require no design change to the system. Use of this tool will be required to examine any middleware and LMMS software produced. Similarly, STEC could use the tool to examine mission unique software and the SYS 500 algorithms could also be examined by Krakatau to gather metrics.

An immediate benefit is the identification of software modules that have questionable quality indicators. Instead of looking at all the software we now can focus on those modules that can be improved. Since some software metrics can be controversial and that there has been no collection of metrics on our software in the past, LMMS will examine previous versions of the middleware to better understand the metrics. Plus LMMS will record system software failures to determine the correlation between failures and our metrics. We must determine which metrics are meaningful to managers and developers alike (it makes sense). Once the metrics are understood then the software will be re-written to meet standards. As a minimum, the recommended metrics to be examined are:

A. Project Metrics

1. Depth of Project class Inheritance Tree (DPIT). This metric measures the maximum depth of the class inheritance tree for the whole project. Large values for DPIT tend to indicate that classes are less specialized and so increase the possibility for reuse.

2. Average Paths (AVPATHS). This metric measures the average number of paths in a call tree. As the value increases testing and debugging become more difficult. Applications with large values of component levels may contain a large number of untested paths that tend to be unexpectedly executed. Often this problem can plague client/server applications.

B. Class Metrics

1. Class Size in Attributes (CSA). This metric measures class size by counting the number of attributes of a class but not the inherited attributes. If a class has a high number of attributes, it may be wise to consider whether it would be appropriate to divide the class into subclasses.

2. Coupling Between Object classes (CBO). This metric is the number of classes which a particular class refers to. CBO is a measure of how cohesive (and therefore reusable) a particular class is likely to be.

3. Lack of Cohesion of Methods (LOCM). This metric measures the percentage of methods that do not access a specific attribute averaged over all attributes in a class. A low percentage indicates high coupling, high testing effort, and potentially low reuse.

4. Average Operation Size (OSavg). This metric measures the average size of the operations (methods) for a particular class. In Object Oriented Development, it is desirable to have small, fine-grained methods. Large values for OSavg are a warning that it may help improve code quality if some methods in the class were broken down.

5. Average Number of Parameters for methods in a Class (NPavgC). This metric provides an average number of parameters for methods in a class not including the inherited methods.

Methods with a high number of parameters generally require considerable testing and are more complex. Thus the code tends to be less maintainable.

C. Method Metrics

1. McCabe's Cyclomatic Complexity (V(G)). This metric measures the number of possible paths through an algorithm. The higher the value the more complex the algorithm and therefore harder to test and maintain.

2. Essential Complexity (eV(G)). This metric is a measure of the structure of testable paths in a component. Values exceeding a value of seven usually signal an unstructured implementation of control logic. This metric indicates difficulty and feasibility of testing or making future code modifications as the presence of unstructured code increases.

3. Maximum Number of Levels (NEST). This metric measures the number of If..Then or If..Then..Elses in a "nest." Units with a large number of "nested" levels may need implementation simplification and process improvement.

4. Halstead Program Vocabulary (n). This metric is a measure of the number of unique operands and operators used in a unit. This can provide an impression of comphrensibility and complexity for that file.

5. Halstead Effort (E). This metric while complex provides an indicator of the difficulty of making a change. Generally, the lower a program's measure of effort, the simpler it is to change the program.

6. Quality Criteria Profile – Maintainability (QCP_MAINT). This metric provides an indicator of the ease in which the code can be maintained. A large value indicates difficulty in maintenance.

7. Quality Criteria Profile – Correctness (QCP_CRCT). This metric provides an indicator of code to correctly execute. A large value indicates less likely to execute as designed (but not necessarily cause a fault).

8. Quality Criteria Profile – Reliability (QCP_RLBTY). This metric provides an indicator of how reliable the software is to not generate a fault. A large value indicates less reliable code.

D. File Metrics

1. Halstead Program Vocabulary (n). Basically the same as the Halstead Vocabulary for the method metrics but applied at the file level. This metric is for quick look ability only.

2. Halstead Effort (E). Basically the same as the Halstead Effort for the method metrics but applied at the file level. This metric is for quick look ability only.

3.2 Mission Dependencies

There are no mission software areas affected and no rework is required. However, certain metrics may induce a desire for changes to the software prior to software acceptance.

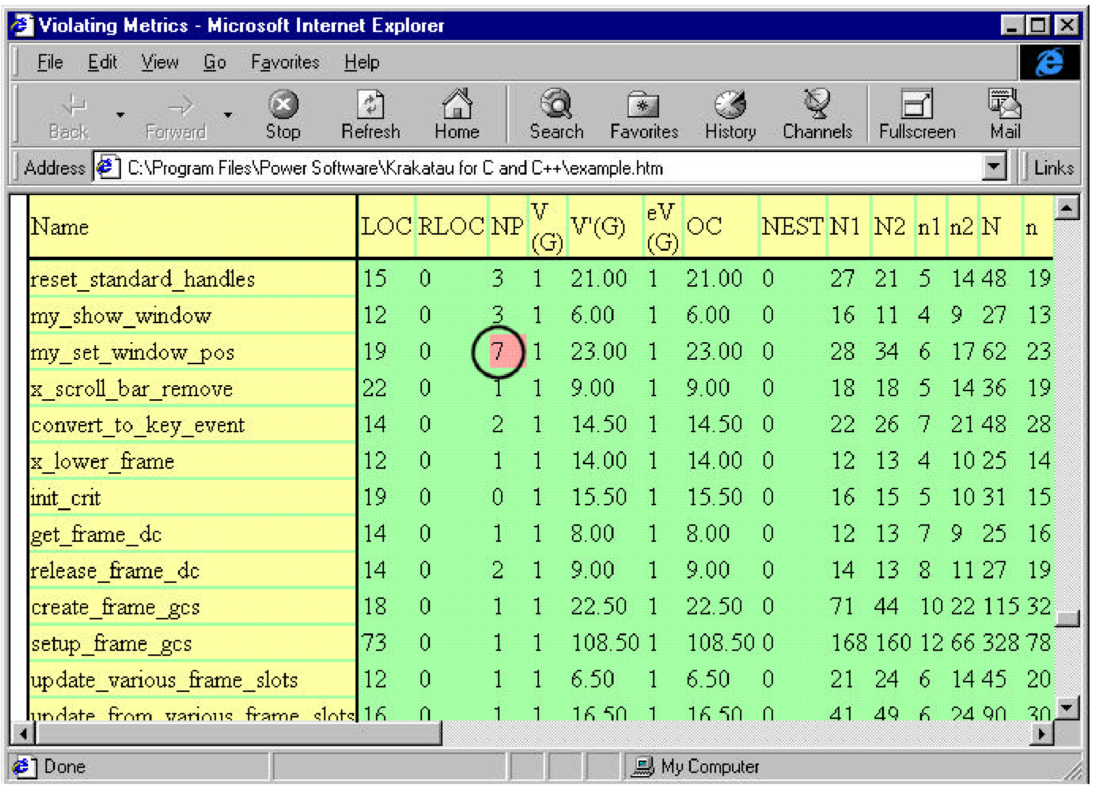
3.3 Core Dependencies

The Krakatau tool will only be required in the Viper lab where software can be examined in a safe manner.

3.4 Benefits

There are several benefits that will be provided by using a metric tool on our software. The major benefit will be an increase in reliability. By examining our software with Krakatau we will better understand the software, reduce maintenance costs, reduce the complexity of the software, and identify existing code that should be re-examined by programmers to improve performance and commenting. Krakatau can also be tied in with the configuration management tools such as ClearCase and Concurrent Version System (CVS).

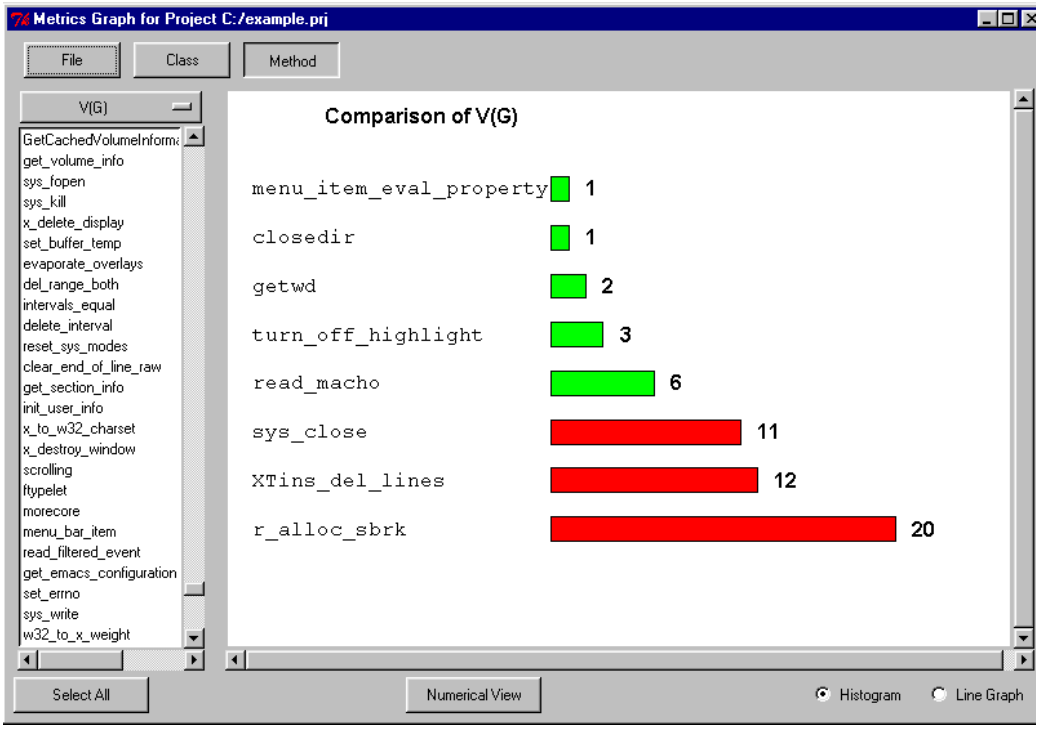
Krakatau provides HTML reports, graphs, an editor and project manager, see Figures 1 through 4 respectively. The product is easy to use, requires about 20 minutes of training, and therefore will be part of every developer's quality process.



Violating Metrics - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address C:\Program Files\Power Software\Krakatau for C and C++\example.htm
Links
Done
My Computer

Name	LOC	RLOC	NP	V (G)	V'(G)	eV (G)	OC	NEST	N1	N2	n1	n2	N	n
reset_standard_handles	15	0	3	1	21.00	1	21.00	0	27	21	5	14	48	19
my_show_window	12	0	3	1	6.00	1	6.00	0	16	11	4	9	27	13
my_set_window_pos	19	0	7	1	23.00	1	23.00	0	28	34	6	17	62	23
x_scroll_bar_remove	22	0	1	1	9.00	1	9.00	0	18	18	5	14	36	19
convert_to_key_event	14	0	2	1	14.50	1	14.50	0	22	26	7	21	48	28
x_lower_frame	12	0	1	1	14.00	1	14.00	0	12	13	4	10	25	14
init_crit	19	0	0	1	15.50	1	15.50	0	16	15	5	10	31	15
get_frame_dc	14	0	1	1	8.00	1	8.00	0	12	13	7	9	25	16
release_frame_dc	14	0	2	1	9.00	1	9.00	0	14	13	8	11	27	19
create_frame_gcs	18	0	1	1	22.50	1	22.50	0	71	44	10	22	115	32
setup_frame_gcs	73	0	1	1	108.50	1	108.50	0	168	160	12	66	328	78
update_various_frame_slots	12	0	1	1	6.50	1	6.50	0	21	24	6	14	45	20
update_from_various_frame_slots	16	0	1	1	16.50	1	16.50	0	41	49	6	24	90	30

Figure 1. Krakatau HTML report.

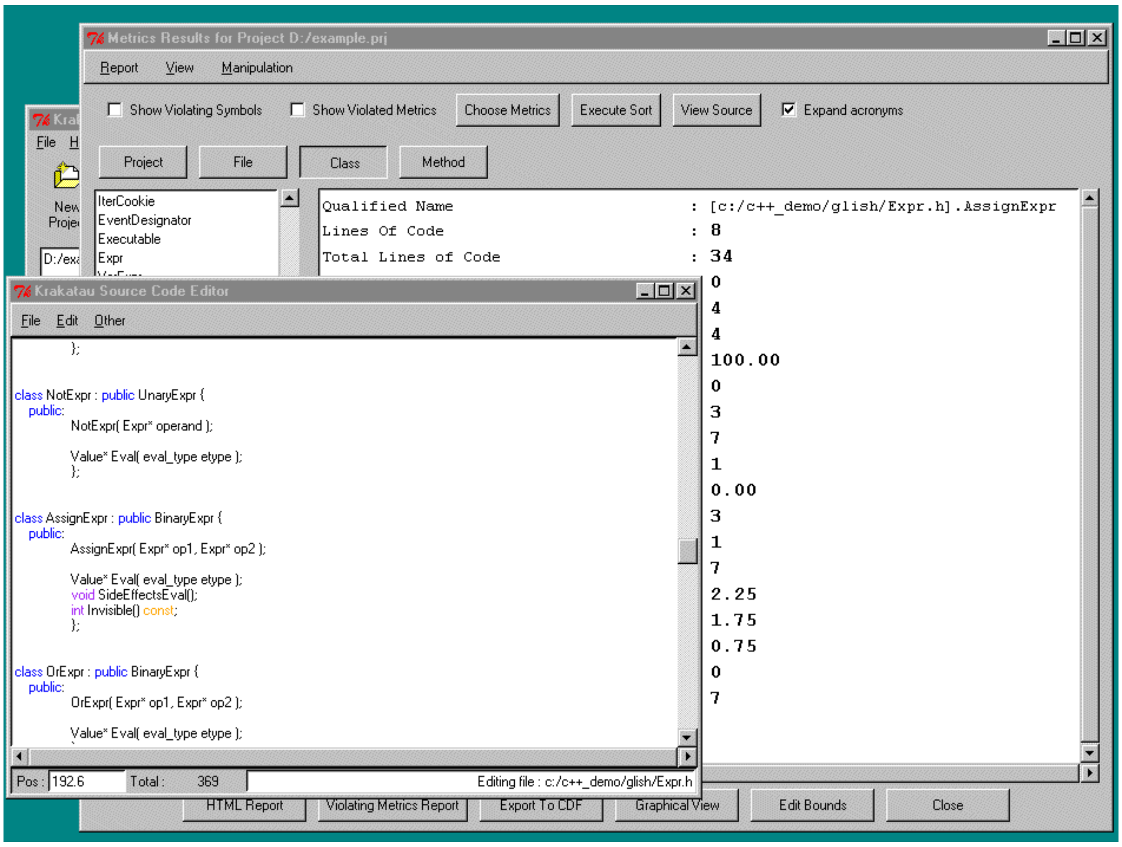


```
GetCachedVolumeInform
get_volume_info
sys_fopen
sys_kill
x_delete_display
set_buffer_temp
evaporate_overlays
del_range_both
intervals_equal
delete_interval
reset_sys_modes
clear_end_of_line_raw
get_section_info
init_user_info
x_to_w32_charset
x_destroy_window
scrolling
ftypelet
morecore
menu_bar_item
read_filtered_event
get_emacs_configuration
set_errno
sys_write
w32_to_x_weight
```

Comparison of V(G)

menu_item_eval_property	1
closedir	1
getwd	2
turn_off_highlight	3
read_macho	6
sys_close	11
XTins_del_lines	12
r_alloc_sbrk	20

Figure 2. Krakatau metrics graph.



```
IterCookie
EventDesignator
Executable
Expr
```

```
Qualified Name          : [c:/c++_demo/glish/Expr.h].AssignExpr
Lines Of Code           : 8
Total Lines of Code     : 34
                          0
                          4
                          4
                          100.00
                          0
                          3
                          7
                          1
                          0.00
                          3
                          1
                          7
                          2.25
                          1.75
                          0.75
                          0
                          7
```

```
        };

class NotExpr : public UnaryExpr {
    public:
        NotExpr( Expr* operand );

        Value* Eval( eval_type etype );
        };

class AssignExpr : public BinaryExpr {
    public:
        AssignExpr( Expr* op1, Expr* op2 );

        Value* Eval( eval_type etype );
        void SideEffectsEval();
        int Invisible() const;
        };

class OrExpr : public BinaryExpr {
    public:
        OrExpr( Expr* op1, Expr* op2 );

        Value* Eval( eval_type etype );
```

Pos : 192.6   Total : 369   Editing file : c:/c++_demo/glish/Expr.h

Figure 3. Krakatau's code editor.

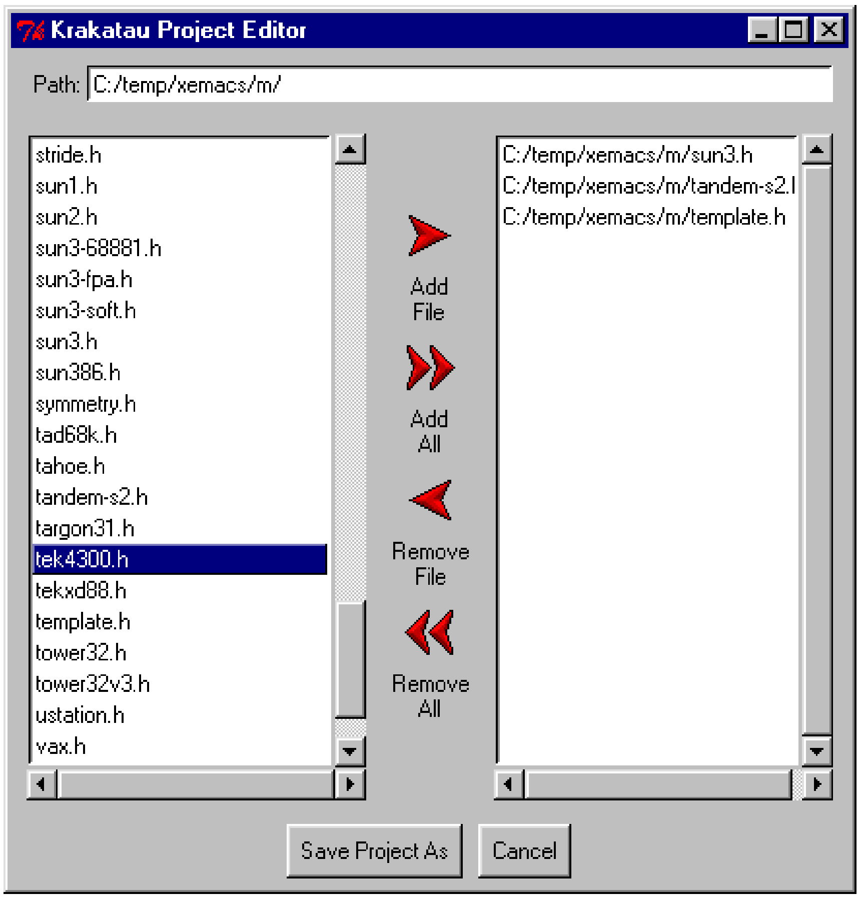
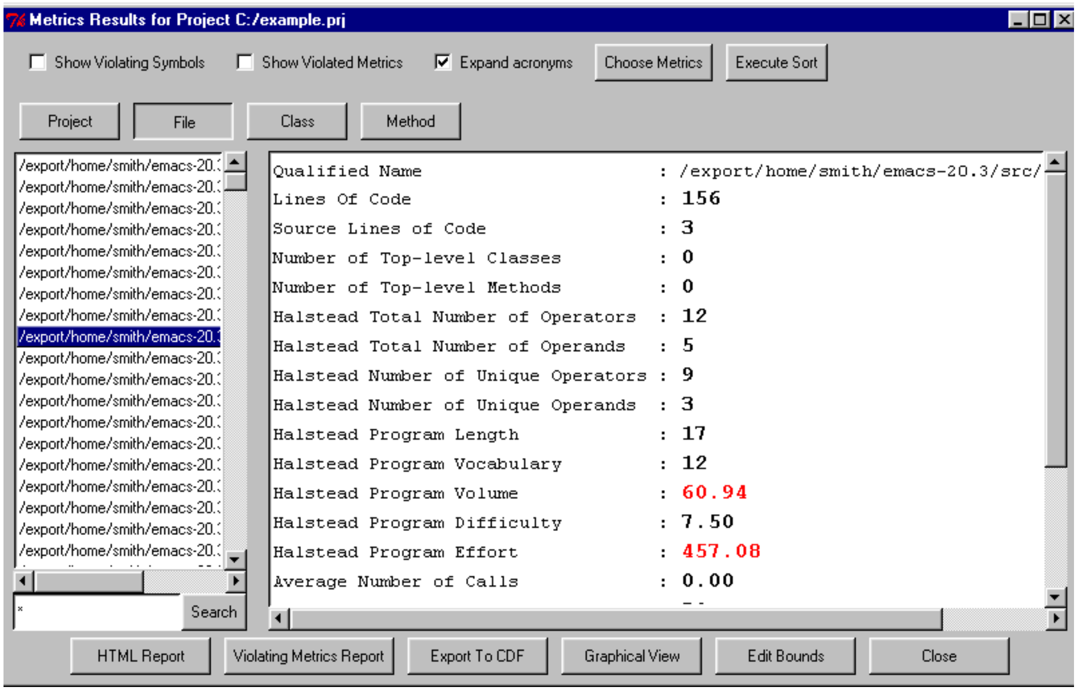


Figure 4. Krakatau's project editor.

Figure 5 shows the main results screen for Krakatau. As can be seen the user can select metrics based on a project, file, class or method level. The user can select "Choose Metrics" to view only to those metrics we are interested in monitoring or the user can select to view only the metrics that are "violating" their bounds. Boundary limits can also be modified to match our methodologies and processes.



```
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
/export/home/smith/emacs-20.3
```

```
Qualified Name                     : /export/home/smith/emacs-20.3/src/
Lines Of Code                      : 156
Source Lines of Code               : 3
Number of Top-level Classes        : 0
Number of Top-level Methods        : 0
Halstead Total Number of Operators : 12
Halstead Total Number of Operands  : 5
Halstead Number of Unique Operators : 9
Halstead Number of Unique Operands : 3
Halstead Program Length            : 17
Halstead Program Vocabulary        : 12
Halstead Program Volume            : 60.94
Halstead Program Difficulty        : 7.50
Halstead Program Effort            : 457.08
Average Number of Calls            : 0.00
```

Figure 5. Krakatau main results screen.

Data can be exported to other products such as Microsoft Excel. As an example of an Exel graph, Figure 6 shows the McCabe Complexity for those software modules in RSCtlmdisp (RSCtlmdisp.c, RSCtlmdisp.h, RSCtlmdispILutil.c, RSCtlmdispILutil.h, RSCtlmdispObjcolor.c, RSCtlmdispObjcolor.h, RSCtlmdisputil.c, and RSCtlmdisputil.h) that have a complexity figure of 20 or higher. This type of graph would allow project managers and developers to instantly see that the code is meeting quality standards.

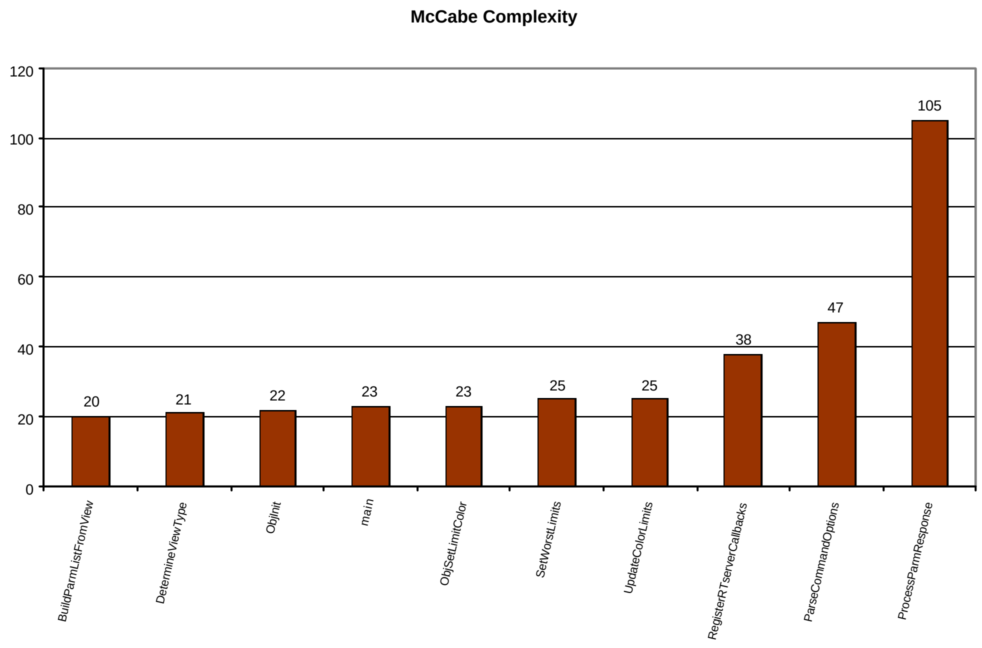


Figure 6. Graphing Example

The next release of Krakatau will provide hierarchical reports, violating metrics reports, and reports with embedded with Java applets for inheritance and cross-reference browsing. Supports printing directly from the user interface and from the reports as the current version of Krakatau will not print at all. As of this date, Power Software is unable to provide the release date.

3.5 Technical Challenges

There are no technical challenges using Krakatau.

3.6 Schedule

There are no impacts to the schedule by acquiring Krakatau.

3.7 Cost Feasibility

To sufficiently support our needs a site license of Krakatau will be required. It is also recommended that annual maintenance and support be purchased to acquire future versions of Krakatau. The annual maintenance cost is 15% of the total license price. Power Software will allow for some the licenses to be used at another site such as CERES. Therefore, software developed at CERES could also be examined using the Krakatau tool.

4.0 COMMANDERS DISPOSITION

	Approved with funding
	Approved as UFR
	Tabled – Rationale should be documented for historical record
	Returned for further work

5.0 REFERENCES

[LAMB1] Lamb, D. A., and Abounader, J. R., *A Data Model for Object-Oriented Design Metrics*, External Technical Report ISSN-0836-0227-1997-409, 1 Oct 97, pg. 1.

[PRESS17] Pressman, R. S., *Software Engineering: A Practitioner's Approach*, McGraw-Hill, 2d ed., 1987, pg. 17.