Evaluation of LOGISCOPE

Julian J. Bunn
February 12th. 1991

Description of Product

Logiscope is a set of software analyzers that determine the quality of software. The code's "quality" is measured in terms of so-called "software metrics", which indicate its complexity, understandability, testability, description and intricacy.
The Logiscope analyzers divide into two basic types; static analyzers and dynamic anaylzers.

The static analyzers produce graphs of values of the "software metrics", and, for example, graphs of the control flow between modules in the source. The metrics are derived from formulae which depend solely on functions of the number of operators and operands used in the code, the number of statements in the code, the number of comment lines, and the number of control branches. The static analyzers thus consist of tools which read in one or more files of source code and produce summary information, and tools which then "edit" that summary information to produce graphs of the interesting metrics.

The potential interest of such metrics to the HEP community is *presumably* for automatic checking by software managers of the quality of submitted code.

The dynamic analyzers offer similar functionality to tools like DEC's PCA, and IBM's IAD, namely that the user obtains information on the run time performance and code utilisation of his software. Since the above tools (PCA and IAD) are generally available, the Logiscope equivalents were not evaluated.

The tools are marketed by VERILOG, a company based in Toulouse, France. The versions evaluated were:

LOGISCOPE - Static and Dynamic Graphic Editor and Archiver—Release 2.1
LOGISCOPE - Static and Dynamic Analyser FORTRAN—Release 1.2

Purpose of Evaluation

There were two aims to the evaluation:

- to evaluate how useful "software metrics" are for managers of large bodies of software,
- to evaluate how well the Logiscope tools work.

Period of Evaluation

The evaluation was made over a two month period beginning in November 1990. The integrated time spent on installation of the Logiscope tools was approximately three days, whereas the integrated time spent using the tools was approximately one week.

Evaluation Platform

The tools were installed on a VAXStation 3100 with 8 MBytes of main memory running VMS 5.4 with the DEC-Windows environment. (With this set-up it was also possible to run the tools with output to a display on a remote station over DECnet.)

Installation

The tools were supplied on a TK50. The installation instructions and general procedure were found to be poor. In particular:

- The instructions in the provided manual regarding mounting the TK50 cassette were unclear and would have been insufficiently detailed for a non-expert.
- The installation command file omitted the /LOG qualifier with BACKUP, so that the installation proceded without informing the installer of what was happening.
- A crucial license file, required before the tools could function, was missing from the installation tape.
- Obtaining a copy of the missing license file from Verilog took a long time, and when the file arrived (as a listing sent by FAX), there were no instructions on the particular way it should be entered in the system.

Customer support from Verilog was very helpful, even if it did not produce solutions quickly!

How the tools Work

One can choose from a variety of programming languages that the tools understand. For this evaluation, the Fortran versions of the tools were used. Given one or more files of code to be analysed, the user first runs a "static analyzer". The static analyzer might be that which understands only standard-conforming Fortran 77, or that which also understands VAX extensions.
The static analyzer produces an intermediate file containing counts of

1. the number of operators,
2. the number of operands,
3. the number of distinct operators used,
4. the number of distinct operands used,
5. the number of comment lines,
6. the number of statements,

for each module in the input file, and globally over all modules. This file is then to be used as input to the "static editor".

The "static editor" is not an editor at all. It is really a "static viewer" which enables the user to plot various types of graphs and tables containing the values of the parameters in the above list. The basis of the Logiscope tools is derived from work done by Halstead on "Software Science", and McCabe on structural complexity of software. For references to this work (unfortunately, after fairly exhaustive search in the Logiscope manuals, not uncovered), please visit the library! "Software Science" is a science that involves the study of program data in terms of textual complexity. Structural complexity pertains to the characteristics of the orientated graphs associated with programs.

In any case, the four essential numbers that are used in various ways by the static editor are:

- n1 : the number of distinct operators
- n2 : the number of distinct operands
- N1 : the total number of operator occurences
- N2 : the total number of operand occurences

The *vocabulary size* is then defined as n1+n2, and the *program length* is defined as N1+N2. Also in terms of the above variables, Halstead defined, for example, the *mental effort* figure, a number proportional to the effort someone would have to make to understand the code.

Perhaps most excitingly, statistical measurements made by Halstead showed that the time taken to code a piece of software was directly proportional to this mental effort number. In fact, *the time needed to code the software, in seconds, was equal to the mental effort figure divided by 18*!

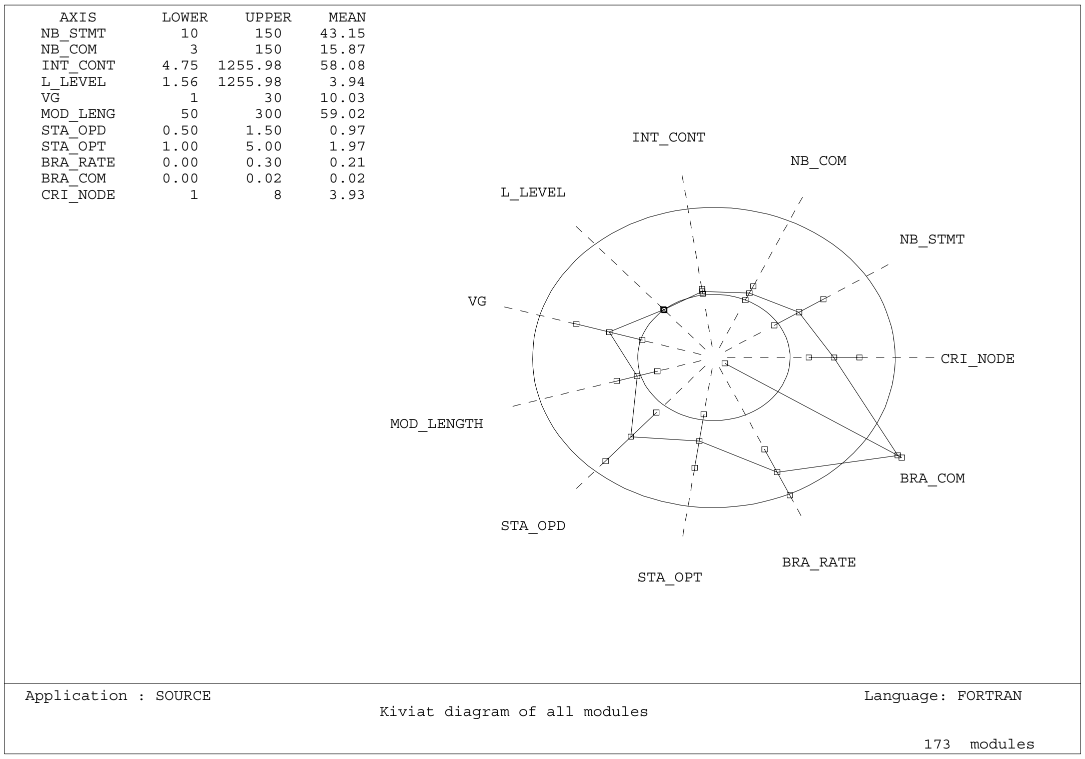
The static analyzer will plot the values of any combination of these variables in various ways. It will plot them per module, per set of modules, or for the whole source code. The user may also define his own formulae for new metrics, and plot those alongside the standard set. I do not want to describe in detail the different graphs and tables available, of which there are many, except to give an example of a *Kiviat Diagram* in Figure 1. (The kiviat Diagram shown is that for JULIA (see Table 1 below).)

The Kiviat Diagram is a handy way of showing the values of more than two variables on the same plot. In the example, eleven different metric values are plotted. Each value is shown as a point along a radius of a squashed circle. The points are joined together to guide the eye; the best result is when all the points lie within the upper and lower limits, shown by the two circles. The associated key to the upper left of the diagram indicates what the lower and upper limits are for each value. The limits are hard-coded by the Logiscope user. For example, for the program analysed, the average Program Length (number of operators plus number of operands) is 59.02. The defined acceptable limits are between 50 and 300. The small squares on each radius correspond to plus and minus one standard deviation from the plotted mean value.

Quality of Documentation and On-Line Help

The user can select at installation time whether he would like a French or English version of the tools. The paper documentation comes in the form of an administrator’s manual, a guide to the static and dynamic analyzers, and a guide to the static and dynamic editors. These guides contain separate descriptions for the various platforms on which the tools can run (Sun, VAX, Apollo, etc.). All the guides lack indexes, so it is usually difficult to quickly find what one is looking for. But they are rather complete and well-written.
In the version of the tools evaluated, there was no on-line help. This is unacceptable.

Figure 1: Example of a Kiviat Diagram



```
   AXIS       LOWER    UPPER    MEAN
 NB_STMT        10      150    43.15
 NB_COM          3      150    15.87
 INT_CONT     4.75  1255.98    58.08
 L_LEVEL      1.56  1255.98     3.94
 VG              1       30    10.03
 MOD_LENG       50      300    59.02
 STA_OPD      0.50     1.50     0.97
 STA_OPT      1.00     5.00     1.97
 BRA_RATE     0.00     0.30     0.21
 BRA_COM      0.00     0.02     0.02
 CRI_NODE        1        8     3.93
```

```
Application : SOURCE                                         Language: FORTRAN
                            Kiviat diagram of all modules
                                                                173  modules
```

Ease of Use

On VMS the tools are accessed using symbols, which involves the user typing commands such as:

```
    AL1W26>  run scope:[com]edsta
```

This is cumbersome; it would be much better to have a VMS verb defined called EDSTA, which could take qualifiers such as /OUTPUT and so on, in the normal VMS way.

As already mentioned, the Fortran versions of the tools were used. There were several unfortunate limitations with the tool that was supposed to understand VAX Fortran extensions, namely:

- the "include" directive was not understood,
- the "implicit none" statement was unrecognised,

- extended source form, where statements are allowed to extend up to column 132, was incorrectly parsed.

Despite these rather serious limitations, the tools worked well and quite quickly, even on a VAX station 3100. The whole of GEANT could be run through the static analyzer in around 15 minutes elapsed time. Unfortunately, several files are likely to be created "behind the user's back", and frequent PURGE commands are necessary to keep disk usage within reasonable bounds! The tools do not inform the user, when running, which files are being created or used.

Some Results

In this section, results are shown of running the tools on a semi-random collection of programs. Programs were selected in one of the following categories: a) being well-known, b) being generally renowned as dreadful code, or c) of interest due to the methods used. These are the programs used (with thanks to the authors for their permission):

- FATMEN : An early version of the File and Tape Management package.(Shiers)
- GEANT : The source from CERN:[PRO.SRC]GEANT.FOR.(Brun)
- CONVERT : A utility for converting Fortran 77 to Fortran 90 (Metcalf)
- JULIA : The steering part of the Aleph Reconstruction Program.(Knobloch)
- ESME : Accelerator particle tracking (remarkably obtuse).
- BAD : A random collection of awful and badly written subroutines.
- SIMUL : A simulation program for SHIFT.(Bunn)

Table 1: Values of Software Metrics for the sample programs

Program	Comment Frequency[1]	Time to Program in Hours[2]	Error Estimate[3]	Abnormal Exits[4]
FATMEN	39%	195[5]	0.54	1.2
GEANT	42%	5	0.95	0.14
CONVERT	39%	6	1.5	0.0
JULIA	49%	1	0.48	0.11
ESME	12%	5	1.0	2.0
BAD	10%	196[5]	13.0	1.25
SIMUL	12%	114[5]	13.0	11.

[1]The Comment Frequency is the percentage of lines in the source form which are comment lines. There is, of course, no account taken of whether the comments are understandable or helpful. In fact, a program containing 500 lines of code and 500 blank comment lines would rate 50% in this column.

[2]The Time to Program in Hours refers to the estimated average time in hours that was needed to code each module in the source.

[3]The Error Estimate is the estimated number of errors in each module in the source.

[4]The Abnormal Exits is the number of abnormal module exits, which increments the number of test cases needed to fully test the source.

[5]These odd figures may be due to either one of the Logiscope variables hitting an upper limit, or one particular module skewing the average over all modules to an artificially high value.

One advantage of the Logiscope tools is that they allow the user to define his own software metrics. The HEP interest might be that a software manager would first pass a file of submitted code through a coding convention checker (such as Floppy), and, that test having been passed, pass the file through the Logiscope tools to decide on its comprehensibility, documentation etc. according to a set of pre-defined limits. (Maybe these tests would be done the other way around.) I thus spent some time trying to decide on a set of metrics which would allow several *HEP values* to be assigned to a Fortran module. This study begged the question, "what makes a good piece of code?", which is difficult to answer. Maybe, a good piece of code is:

- Concise : there are few statements. Each statement refers to several of the declared variables in the module.
- Simple : there are few decisions (IF-THEN-ELSE constructs).
- Well Described : a good fraction of the source is comment lines. (But a simple program needs fewer comments.)

Logiscope allowed me to express formulae for these three categories in terms of the given variables (n1, n2, N1, N2, number of statements and number of comments), and then plot the Kiviat Diagrams showing the metric values for each of the sample codes. In table format, the results were as follows (the bigger the number, the better):

Table 2: HEP Value for the sample programs

Program	Concision	Simplicity	Description
FATMEN	1.5	9.1	159
GEANT	1.2	5.0	65
CONVERT	0.98	4.2	67
JULIA	1.5	5.5	61
ESME	1.3	4.3	9
BAD	0.40	3.8	120
SIMUL	0.60	3.4	224

Given the above table, one can see how to speed up checking of submitted code. An acceptable range is defined in each of the categories, and the code is said to be "Accepted" if it falls well within the limits, "Critical" if it falls close to either limit, or "Rejected" if it falls outside the limits. (For these metrics, the upper limits should all be set rather large.) I chose lower limits of:

- Concision : 1.0
- Simplicity : 5.0
- Description : 100.

with the following results:

Table 3: HEP Acceptability of the sample programs

Program	Concision	Simplicity	Description
FATMEN	accepted	accepted	accepted
GEANT	accepted	critical	rejected
CONVERT	critical	critical	rejected
JULIA	accepted	accepted	rejected
ESME	accepted	critical	rejected
BAD	rejected	rejected	accepted
SIMUL	rejected	rejected	accpeted

Conclusions

1. "Software Metrics" provide useful indicators of code quality. It is unclear whether the information they give is *reliable* enough to substitute a thorough examination of the source code by an experienced programmer. It is also unclear *which* of the many possible metrics should be selected as the best indicators of code quality. Some of the metrics give an idea of how complex the code is, some how easy the code is to test. The whole area needs a careful and in-depth study before proper conclusions can be drawn. Certainly, this evaluation has barely scratched the surface.

NOTE

It should be stressed most strongly that the potential benefits of software metrics for HEP code management need detailed study.

2. The Logiscope tools have an old-fashioned user interface, with no on-line help, and their installation is error-prone. However, they function well and quickly, and offer to the user a plethora of ways of displaying the values of software metrics. In this evaluation, the Fortran versions of the tools were used, but versions exist for other high-level languages. So that, in principle, HEP code quality tests could be made independently of the implementation language.

Disclaimer

Some would argue that the evaluation period of two months is too short for serious conclusions to be drawn. A counter argument is that, if such tools are to be used by busy software coordinators, then any tool that takes a long time to master is either a) too complicated, or b) too poorly documented. In view of possible diagreement over the merit of this evaluation, I would like to stress that these are my opinions of the Logiscope tools, and mine alone. I would encourage anyone interested in software metrics to investigate such tools themselves, and not take my word alone! Also, I understand that a more recent version of the tools is available which includes a better user interface.