Measuring the Quality of Functional Programs: an Empirical Investigation

R Harrison, L G Samaraweera, M R Dobie, P H Lewis
Dept. of Electronics and Computer Science,
University of Southampton, SO17 1BJ, UK

March 13, 1995

This paper describes an investigation into measuring the quality of functional programs. The work reported here is part of a larger, on-going study into a quantitative analysis of the effect of utilizing different programming paradigms on code quality. Prior to undertaking such a comparative analysis it is necessary to establish a baseline of quality indicators which can then be used as metrics for the remainder of the project. Thus the aim of the research presented here was to evaluate a set of suggested indicators corresponding to internal attributes by investigating the correlation between the suggested indicators and the desired external quality-type attributes of the code. A method for the evaluation of suggested metrics is discussed and the results of performing such an evaluation for functional programs are presented.

Keywords: quantifying quality, functional languages, internal attributes

INTRODUCTION

The research described in this paper was performed as part of a project concerned with investigating the variations in code quality resulting from the use of different programming paradigms. In particular, the initial aim of the project was to investigate whether or not the quality of code produced using a functional language was significantly different from that produced using an object-oriented language [1]. In order to carry out the experiment it was first necessary to determine which metrics should be considered. An earlier paper [2] analysed a variety of metrics and discussed their suitability for measuring functional programs. This paper describes a formal experiment which was set up to evaluate the suggested metrics, and reports on the results of the experiment.

The method described here was influenced by work reported in the literature [7] and also by the work of the DESMET project [13]

METHOD

To evaluate different programming languages it is necessary to measure the external quality-type attributes of the code, such as reliability, usability, maintainability, testability, reusability, integrity, efficiency, and portability. However, with the exception of efficiency, such attributes are notoriously difficult to quantify, because they depend on the way in which the software reacts with external factors, such as developers and users. Metrics based on internal attributes are often employed in the belief that there is a strong correlation between internal attributes and the desired external attributes. Internal attributes such as length, modularity, reuse, coupling and cohesion are much easier to measure than external attributes, and some work has already been done on correlating internal and external attributes for programs written in imperative languages [3, 6, 7]. The experiment described here used established statistical techniques to ascertain whether or not there is any correlation between a selection of internal attributes (based on length and modularity) and certain characteristics of the development process, such as the number of errors found during development, which are assumed to be indicative of quality.

If a significant correlation is found between the internal attributes and the external quality type attributes, then the internal attributes can be used to determine the quality both of software produced in the future and also of existing software, thus greatly assisting quality assurance in software engineering projects.

The internal attributes which were measured are outlined in [2]. We will refer to these as *suggested indicators*, or SIs. These include both simple size measurements, such as the number of non-comment source lines, and measures of modularity such as the number of distinct functions called, the number of distinct domain-specific functions called, and so on. The SIs were chosen partly on the grounds that they are relevant to both functional and object oriented programs and partly because of the ease with which they can be collected.

There are certain characteristics of the code which, under carefully controlled experimental conditions, can be monitored and measured during its development [7]. These characteristics, which we will take to be indicative of the desired external quality-type attributes, will be referred to as *development metrics*, or DMs. They include the following:

- the number of known errors found during testing (KEs)
- the time to fix known errors (TKE)
- the number of modifications requested (MRs)
- the time to implement modifications (TMR)

- a subjective assessment of complexity, provided by the system developers (SCs).

If a significant correlation is found between the suggested indicators and the development metrics, then it will be possible to use the suggested indicators for the quality assurance of functional programs.

The question which will be addressed in this paper is:

> How well do the suggested indicators correlate with the development metrics ?

In order to answer this question, the necessary statistics were collated and analysed for a number of programs.

It was essential that the development cycle was completed for each program, as this ensured that the environment could be monitored closely; the known errors could be logged, together with modification requests and the time needed to implement both. Reviews of analysis, design and test documentation together with code reviews and walkthroughs were all carried out regularly. All code was subjected to rigorous testing prior to the extraction of the metrics.

The language under investigation, SML, was used during the design, implementation and testing of the programs. SML was chosen because of the availability of a high quality, publicly distributed compiler. A functional style of programming has been adhered to as far as it was practical to do so.

The results of this experiment may be affected by a number of independent variables. In particular, the results may be confounded by the following:

- the experience of the research assistants with the application
- the experience of the research assistants with the language used
- the application domain
- the systems and hardware used

In order to reduce the effect that these variables may have had on the outcome of the experiment, certain decisions were taken: only one researcher was employed, to alleviate the problems of different experience levels within the project; the application domain was restricted (see the following section), and the development environment was kept constant throughout the project.

A similar experimental method to this has been used before [7] for quality indicators based on information flow metrics.

THE APPLICATION DOMAIN

In the current project we have specified, designed and implemented a number of image analysis algorithms in SML. The image analysis domain was chosen

because it offers a particularly wide variety of distinct problems for software engineering and we already had expertise in the development of image processing algorithms in C [8]. The subset of algorithms examined so far has included low level algorithms which operate on an image array and produce image arrays as output (for example, convolution algorithms) and intermediate level algorithms which operate on image arrays and produce symbolic output (for example, curvilinear feature extraction algorithms).

As mentioned earlier, similar experiments to these have been performed for imperative languages within different application domains [3, 6, 7]. The programme of work described here has been influenced by these experiments, particularly those partially funded by the Alvey Software Reliability Modelling Project and the ESPRIT REQUEST project [7].

DATA COLLECTION

The attributes were all measured quantitatively: measurements of known errors and modification requests were collated during development by noting each one at the head of the module in which it occurred. Similarly, the times to fix the known errors and modification requests were found by recording the real time before starting and after finishing work on each one. Timings were measured in minutes. The subjective complexity was given by the programmer responsible for the program using an integer from 1 to 5. Software tools for extracting some of the suggested indicators automatically were developed. The suggested indicators were collated after final testing and acceptance of the delivered code.

The following development metrics (DMs) were collected for every program.

- the number of known errors found during testing (KEs).
- the time to fix known errors (TKEs), measured in minutes.
- the number of modifications requested (MRs). This represents the number of changes which were requested excluding changes for fault clearance.
- the time to implement modifications (TMRs), measured in minutes.
- a subjective assessment of complexity (SC), provided by the system developer. This is based on an ordinal integer scale from 1 to 5, where 5 represents the most complex code.

The development metrics were recorded by noting each one in a comment at the head of the module.

The suggested indicators (SIs) which were used are as follows:

- ncsl: the number of non-comment, non-blank source lines. This was collected automatically after delivery of the source code.
- the number of distinct functions, N*, which are called by the program. Each function is only counted once, no matter how many times it is called.

- the number of distinct library functions, L, which are called by the program. In this context, a library function is a general purpose function whose application is not restricted to the image processing domain, for example *map*, *fold*, etc.
- the number of distinct domain-specific functions, D, called by the program, where D = N* - L.
- the depth of the hierarchy chart, depth. (A hierarchy chart [2] is an abstraction from the traditional structure chart [9], and is similar to a call graph).
- the number of function declarations, dec, which are specified within a program. This represents the size of a program's public interface.
- the number of function definitions, def, which are coded within a program. This represents the number of functions which have been implemented specifically for one program.

Note that the counts for N*, D and L will be greater than or equal to the number of function definitions, def, since the latter is restricted to the functions which are implemented within one particular program. Also, def should be greater than or equal to dec.

Twelve sets of image analysis algorithms were developed, together with sets of general purpose library functions. Altogether 176 functions were defined, consisting of a total of 1,595 non-comment source lines. The implementation took 72 hours of connect time (staff hours) and testing required a further 39 hours.

The code was tested rigorously through the use of test scripts with assertions. Only when all known errors were fixed was the code accepted as finished. The SIs were produced by static analysis of the source code following this final acceptance testing.

ANALYSIS AND RESULTS

The aim of this analysis was to determine whether or not there is a significant correlation between the development metrics and the suggested indicators. To measure the relationships between the DMs and the SIs, graphical assessments of the relationships were performed together with correlation analysis using tests of significance at the 5% level.

Data collected from software development projects may be skewed (usually towards zero) and include large values which lie outside the expected sample range. It is also impossible to state that the data has been drawn from a random sample of equivalent items [4, 7] and consequently robust summary statistics must be used when analysing the distribution of the values. This form of analysis is discussed in the following section.

Robust summary statistics

The usual method for investigating the distribution of a set of values is to use a box plot [4, 7, 10]. This displays the median and quartiles of a set of values. A box is drawn from the lower quartile to the upper quartile; thus half of the data will lie within the box. Two theoretical tails are also calculated, representing the range within which the remaining values are expected to fall. The upper tail is found by adding (1.5 * box length) to the upper quartile, and the lower tail is found by subtracting the same amount (1.5 * box length) from the lower quartile. The theoretical tail values are truncated to the nearest value in the set, to ensure that all the calculated values are representative. If the data is not skewed, then the result is a symmetrical box plot with a median placed in the centre of the box and 2 tails which are equal. However, if the data is skewed to the left then the median will be very close to the left hand side of the box and the lower tail will be shorter than the upper; the mirror image of this indicates a set of data which is skewed to the right.

The box plots for the development metrics and the suggested indicators are shown in Figure 1 and Figure 2 respectively. The box plots for the development metrics show that the plots are skewed to the left, indicating that there are a large number of small values in the data sets. An outlier is an unusual or unexpected value, possibly indicating an error. A *possible* outlier is shown by an asterisk, and is defined here as a value which lies between the upper tail and a point (3 * box length) above the upper quartile. A *probable* outlier is represented by a '0' and represents a value greater than (3 * box length) beyond the upper quartile. A probable outlier is thus a stronger indicator of a spurious value than a possible outlier. Outliers can be seen on four of the five box plots.

Turning to the suggested indicators, we find that the box plots for the number of distinct functions, N*, the number of distinct library functions, L, the number of function declarations, decs, and the number of function definitions, defs, are skewed to the left, whereas the box plots for the number of noncomment source lines and the depth of the hierarchy chart are skewed to the right. Two probable outliers can be seen on the plot for the number of declarations, decs; possible outliers can also be seen on three of the other plots.

Since the box plots are skewed and there are some outliers in the data, parametric tests such as Pearson's correlation coefficient must be treated with caution. Consequently two nonparametric correlation coefficients (Spearman's and Kendall's) were calculated as well as Pearson's, as described in section 6.3.

Inspecting the relationships between attributes

The relationships between attributes can be inspected graphically by plotting corresponding indicator-characteristic pairs on scatter diagrams [4]. This gives an immediate indication of any possible relationship and any extreme values.

The scatter diagrams of the development metrics against the suggested in-

dicators were plotted; see, for example, Figure 3. On inspection it was apparent that a correspondence between high values of the number of modification requests and high values of the suggested indicators might exist, with the exception of the number of function declarations. A similar trend is detectable for the time to complete the requests, again with the exception of the number of function declarations. The majority of programs had values close to zero for the number of function declarations and consequently it is very difficult to infer anything from these plots.

Possible relationships between the subjective complexity and the suggested indicators could also be detected, although the subjective complexity often appearing to be relatively small, possibly reflecting the subjectivity of the measure.

The scatter plots for the known errors and the time to fix them, however, showed more random distributions which gave no indications of any clear relationships.

Measuring the relationships between attributes

In order to determine the significance of the relationships between the attributes, a number of different correlation coefficients were calculated. Pearson's correlation coefficient is perhaps the most commonly used measure of correlation. However, it requires certain normality criteria to be satisfied and so should not be relied on as the sole measure of correlation when analysing data which has been collected during software development. Consequently, nonparametric statistical tests such as Spearman's and Kendall's correlation coefficients should be used in addition to Pearson's.

We calculated Pearson's, Spearman's and Kendall's correlation coefficients for pairs of DM, SI values, and used 1-tailed significance tests for significance at the 5% level.

Pearson's correlation coefficients are shown in Table 1. The suggested indicator most closely correlated with the DMs is the number of non-comment source lines, which is correlated with 3 of the five DMs. Of the remaining suggested indicators, all but depth are correlated with both the number of modification requests and the time to do the modifications, with significance at the 5% level.

The Spearman rank correlation coefficient [11, 12] is based on ranks. It requires the variables in the set of data to be measured on at least an ordinal scale. The calculation involves listing the programs and then finding the ranks for the 2 attributes under investigation (for example, known errors and ncsl). The differences between the 2 ranks are then found and used in Spearman's formula. A correction must be made if a large number of ties occurs in the input data; this adjustment was necessary because of the large number of tied values in the data. The significance of the correlation coefficient can be checked by finding the corresponding Student's t statistic. The results of the correlation analysis are shown in Table 2.

It can be seen that there is a significant correlation at the 5% level between some of the suggested indicators and some of the development metrics. Four of the suggested indicators (the number of non-comment source lines, the number of distinct functions called by the program, the number of domain-specific functions called and the number of function definitions) were found to be correlated significantly with the number of modifications, the time to do the modifications and also subjective complexity, with significance at the 5% level.

The depth of the hierarchy chart, however, is only correlated with subjective complexity at the 5% level. Similarly, the number of function declarations only appears to be correlated with the time to fix the modification requests at the 5% level, and this correlation is negative. These findings confirm the implications which can be drawn from the scatter plots by observation.

The Kendall rank correlation coefficient [11, 12] is another measure of association for data which has at least an ordinal scale imposed on it. The values are ranked in increasing (or decreasing) order and the test checks for any correlation between the 2 sets of ranked values. Tied values can also be taken into account by using a modified formula. Kendall's correlation coefficients are shown in Table 3.

The patterns of correlation are very similar to those found for Spearman's correlation coefficients. The results for the number of non-comment source lines, the number of distinct functions called, the number of distinct domain-specific functions called and the number of function definitions are repeated, as is the negative result for the number of function declarations. The number of library functions called was also shown to be correlated with the number of modification requests, the time to fix the requests and the number of known errors, at the 5% level.

Taking the results of all the correlation coefficients into account, it can be seen that the suggested indicators with the strongest correlation is the number of non-comment source lines. The number of distinct functions called by a program, the number of domain-specific functions called and the number of function definitions are also strongly correlated with the development metrics. Out of these, the number of non-comment source lines and the number of function definitions are the simplest to collect through the use of automated processing.

We can also conclude that the majority of the SIs are strongly correlated with the number of modification requests and the time taken to do the modifications. The subjective complexity measure is also closely correlated with the SIs.

The strength of the association between each development metric, suggested indicator pair was assessed using the Fisher exact probability test [12]. This test is used in preference to $2*2$ contingency tables if the number of observations is small, and determines the probability that the observations would have occurred if there was no association between the attributes in question. The full results are shown in Table 4. Several of the programs with high values of the suggested indicators were associated with high development metric values, at the 5% level. For example, 2 of the 3 programs with high values for the number of non-

comment source lines also had a high number of modification requests. The final row shows the number of programs with at least one high development metric value.

CONCLUSIONS

We have detected a significant correlation (at the 5% level) between certain suggested indicators and several of the development metrics which are indicative of software quality. The suggested indicator which was found to have the strongest correlation was the number of non-comment source lines. The number of distinct functions called by a program, the number of domain-specific functions called and the number of function definitions were also found to be strongly correlated with the development metrics. Thus it may be possible to use these suggested indicators to monitor the quality of functional programs in future software projects, and to assess the quality of existing code written in functional languages.

We have also found that the majority of the suggested indicators are closely correlated with the number of modification requests made during the development process, the time taken to complete the modification requests and a measure of subjective complexity. Consequently it may be possible to use the suggested indicators to determine which modules will be most prone to change, or are thought to be the most complex. In particular, it may be possible to use the suggested indicators which are available at design time (such as the number of function definitions) to discriminate between different designs with the aim of maximising quality.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the support of the University of Southampton Research Committee for a grant to undertake this work.

References

[1] **Harrison, R, Samaraweera, L G, Dobie, M R, Lewis, P H**, 'Comparing Programming Paradigms: an Evaluation of Functional and Object-Oriented Programs' submitted for publication (July 1994).

[2] **Harrison, R**, 'Quantifying internal attributes of functional programs' Journal of Information and Software Technology, Vol 35 No 10, (October 1993) pp 554-560

[3] **Fenton N, Melton, A**, 'Deriving Structurally Based Software Measures' JSS 12, (1990) pp 177-187

[4] **Fenton, N E**, Software Metrics, A Rigorous Approach, Chapman & Hall, (1991)

[5] **Fenton, N E**, 'The Mathematics of Complexity' in Computing and Software Engineering, in J. H. Johnson, M.J. Loomes, (eds), Proceedings, The Mathematical Revolution Inspired by Computing, Brighton, Oxford Clarendon Press, (1991) pp 243-256

[6] **Shepperd, M J**, 'Design metrics: an empirical analysis' Software Engineering Journal, Vol 5 No 1, (1990) pp 3-10

[7] **Kitchenham, B A Pickard, L M, Linkman, S J**, 'An evaluation of some design metrics' Software Engineering Journal, Vol 5 No 1, (1990) pp 50-58

[8] **Dobie, M R, Lewis, P H**, 'Data Structures for Image Processing in C' Pattern Recognition Letters, 12, (August 1991) pp 457-466

[9] **Page-Jones, M**, The Practical Guide to Structured Systems Design, Yourdon Press (1980)

[10] **Hoaglin, D C, Mosteller, F, Tukey, J W**, Understanding Exploratory Data Analysis, Wiley, (1983)

[11] **Siegel, S**, Nonparametric statistics for the behavioural sciences, McGraw Hill (1956)

[12] **Siegel, S, Castellan, N J**, Nonparametric statistics for the behavioural sciences, 2nd ed, McGraw-Hill (1988)

[13] **Law, D**, DESMET methodology: overall specification, DESMET Project Deliverable 2.1, National Computing Centre (November 1992)

Suggested	*Development Metrics*				
Indicators	*KE*	*TKE*	*MR*	*TMR*	*SC*
ncsl	0.32	0.00	0.63*	0.54*	0.75*
*N**	0.07	-0.18	0.89*	0.92*	0.34
D	-0.04	-0.27	0.91*	0.92*	0.29
L	0.40	0.17	0.52*	0.66*	0.37
depth	0.28	0.00	0.44	0.42	0.62*
dec	-0.33	-0.39	0.90*	0.92*	-0.11
def	-0.04	-0.27	0.93*	0.85*	0.48
* significant at the 5% level					

Table 1: Pearson's rank correlation coefficients

Suggested	*Development Metrics*				
Indicators	*KE*	*TKE*	*MR*	*TMR*	*SC*
ncsl	0.29	0.00	0.64*	0.58*	0.60*
*N**	0.34	0.00	0.74*	0.79*	0.63*
D	0.26	-0.23	0.68*	0.68*	0.66*
L	0.47	0.27	0.61*	0.74*	0.45
depth	0.33	-0.06	0.27	0.26	0.60*
dec	-0.43	-0.51*	0.26	0.28	-0.08
def	0.24	-0.13	0.67*	0.70*	0.60*
* significant at the 5% level					

Table 2: Spearman's rank correlation coefficients

Suggested	*Development Metrics*				
Indicators	*KE*	*TKE*	*MR*	*TMR*	*SC*
ncsl	0.22	-0.02	0.53*	0.41*	0.54*
*N**	0.22	0.02	0.56*	0.60*	0.56*
D	0.14	-0.15	0.47*	0.49*	0.62*
L	0.37*	0.22	0.40*	0.53*	0.35
depth	0.22	0.00	0.16	0.16	0.52*
dec	-0.35	-0.39*	0.19	0.22	-0.08
def	0.14	-0.06	0.55*	0.52*	0.56*
* significant at the 5% level					

Table 3: Kendall's rank correlation coefficients

No. of progs. with high DM values		*No. of progs. with high values of*						
		ncsl	*N**	*D*	*L*	*depth*	*dec*	*def*
	TOTAL	3	2	3	3	1	2	3
KE	2	0	0	0	1	0	0	0
TKE	3	1	1	1	1	0	0	1
MR	2	2*	1	2*	1	0	2*	2*
TMR	3	2	1	2	2	0	2*	2
SC	2	2*	1	2*	1	0	1	2*
At least 1 high value	6	3	2	3	3	0	2	3
* significant at the 5% level								

Table 4: Number of programs with high DM and SI values

Figure 1: Box plots for the development metrics

Figure 2: Box plots for the suggested indicators

Figure 3: Scatter diagrams for modification requests against suggested indicators