AN EVALUATION OF SOFTWARE STRUCTURE METRICS

Barbara A. Kitchenham
Centre for Software Reliability
City University

ABSTRACT

This paper attempts to evaluate some software design metrics, based on Henry and Kafura's information flow metrics ([1], [2]), using data from a communications system.

The ability of the design metrics to identify change-prone, error-prone, and complex programs was contrasted with that of simple code metrics. It was found that the design metrics were not as good at identifying change-prone, fault-prone and complex programs, as simple code metrics (i.e., lines of code and number of branches).

It was also observed that the compound metrics, built up from several different basic counts, can obscure underlying effects, and thus make it more difficult to use metrics constructively.

INTRODUCTION

The main purpose of this study was to investigate the practical use of design metrics, and in particular, the information flow metrics developed by Henry and Kafura ([1], [2], [4]). These metrics measure the links among procedures in terms of the flow of information among procedures. Information flow metrics are relevant to any system developed using a structured design technique, and any system which can be represented in terms of a structure chart ([6], [8]). Information flow metrics measure:

- the links among procedures caused by procedure calls,
- the links among procedures and common data structures as a result of reading to and writing from such structures,
- the links among procedures as a result of passing data in the parameters of procedures.

Henry and Kafura found that information flow metrics were able to identify change-prone UNIX procedures and evaluate potential UNIX design changes. The aim of this study was to establish whether the metrics could be validated in another environment. Confirmation that these metrics are of general use is important because they could be of great practical value to software engineers and managers, since they allow:

- objective evaluation of a system's design structure, with the opportunity to indicate and evaluate alternative structures,
- identification of change-prone and error-prone procedures early enough in the design process to ensure adequate resources are given to them (e.g., assignment to experienced developers for implementation, provision for extra testing, or more stringent testing criteria, etc.),
- an improved ability to cost system enhancements, by identifying a set of procedures to be changed as difficult, average, or simple with respect to information flow metrics in order to obtain weights for cost estimation (particularly for detailed design and integration testing).

The evaluation procedure used in this study was to investigate the relationship between the design metric values for individual programs (i.e., the basic code units developed by the software developers) and:

- the number of changes made to the program as a result of subsequent system enhancements,
- the number of changes made to the program as a result of program faults,
- a subjective assessment of program complexity provided by the team leader of the system developers.

The ability of the design metrics to identify change-prone, error-prone, and/or complex programs was compared with that of simple code metrics, in order to assess the relative usefulness of the design metrics.

DATA COLLECTION

Data collection was done by manual inspection of the code and documentation of each program in the communications system. Data collection was based on the program units as they were developed and maintained by the software development staff. Thus, some of the programs, such as subroutines, were not directly compilable entities and did not have the "clean" procedural interface assumed by Henry and Kafura. For example, some subroutines manipulate the on-stack variables of their parent program rather than return explicit parameter values.

For each program in the system, the following quantitative data was collected:

369
1. the number of programs which call a specified program (CP),
2. the number of programs called by a specified program (PC),
3. the number of input parameters on the program interface (IP),
4. the number of output parameters on the program interface (OP).
5. When parameters are used as both input and output, they are included in both counts,
6. the number of data structures (not individual elements) the program reads from, but does not also write to (RD),
7. the number of data structures the program writes to, but does not also read from (DW),
8. the number of data structures the program both reads from and writes to (DB),
9. program size in lines of code, i.e. non-comment, non-blank lines, in program (LSE),
10. program control flow measured in terms of the number of branches, (CF). Notional branches were included so that IF-THEN-IF and IF-THEN-ELSE-IF were both counted as 2 branches. The number of branches for loops with a single control structure (i.e. FOR, WHILE, or UNTIL) was counted as 2, and for loops with a dual control structure (i.e. FOR and WHILE, or FOR and UNTIL) was counted as 3. The compiler evaluated compound booleans lazily, so each AND and OR in a conditional statement or loop control was counted separately.
11. program enhancements, i.e. the number of times the program was amended excluding changes for fault clearance (GCH).
12. fault counts in each program which recorded each change to the program during its development and subsequent evolution.
13. number of known errors, i.e. the number of faults corrected in the program (KE).
14. This information was obtained from formatted comments in each program which recorded each change to the program during its development and subsequent maintenance. Faults found during development and modifications were recorded (i.e. faults detected by code reading, unit, integration and system testing) as well as faults detected in released versions of the system.
15. subjective complexity, i.e. a rating of the complexity of the program on a scale of 1 (very simple) to 5 (very complex) provided by a member of the development group (SC). The same person performed all the ratings.

A complete set of values was obtained from a total of 226 programs. There are two problems with the data collection procedure. Firstly, manual collection of data is itself error-prone.
1. Modified informational fan-in of a procedure (MIFI) =
   number of procedures which call the procedure (CP)
   + number of data structures from which the procedure receives data (DW + DB)

2. Informational fan-out of a procedure (IFO) =
   number of procedures called by the procedure (PC)
   + number of output parameters (OP)
   + number of data structures into which the procedure places data (OW + DB)

The formula for procedural complexity, also, caused problems because a number of programs had zero values for MIFI or IFO, and would, therefore, have had a zero procedural complexity. This was partially a result of including subprograms which did not exhibit conventional interfaces, but is also a general difficulty for system interface programs which are called directly by system users, or by other independent systems. Since a summation of fan-in and fan-out values preserves the ranking obtained by multiplying and squaring the fan-in and fan-out values, and avoids the problem of programs being spuriously assigned zero complexity values, the following synthetic was used as an overall information linkage metric (IL):

\[ IL = MIFI + IFO \]

The linkage metric was not multiplied by program size, as it should have been in order to simulate the procedural complexity metric more closely, because for both evaluation and interpretation purposes, Henry and Kafura consider the information flow part of their procedural complexity metric separately from the code size part.

DATA ANALYSIS AND RESULTS

Distribution of metric values

The distribution of data values for the metrics IL (information linkage), LER (program size), CHG (changes), KE (known errors), CF (control flow in terms of number of branches), SC (subjective assessment of complexity) is shown in figure 1 as a set of frequency histograms, which show the number of programs with very small, small, average, large and very large values for each metric.

Relationships between metrics

This section reports on analyses aimed at investigating relationships between quality indicator metrics (i.e. information linkage, IL, control flow, CF, and size, LER) and the quality characteristic metrics (i.e. changes, CHG, known errors, KE, and subjective complexity, SC).

The initial analysis investigated whether programs with large values of the quality indicator metrics corresponded to programs with large values of the quality characteristic metrics. The programs with large and very large metric values (as shown in figure 1) were identified and tabulated to show the degree of overlap between programs with a large value (i.e. large or very large) of one or more quality indicators and programs with a large value of one or more quality characteristics.

Of the 226 programs investigated:

- 135 exhibited no large values of either the quality indicator metrics or the quality characteristics.
- 12 had at least one large value of an indicator metric, but average or small values for all the quality characteristic metrics.
- 47 had average or small values for all the indicator metrics but a large value for at least one of the quality characteristic metrics.
- the remaining 32 programs had large values for at least one of the indicator metrics and at least one of the quality characteristic metrics.

In practical terms, this means that if programs with large values of any one of the three indicator metrics had been selected for special treatment during initial product development (e.g. extra development time, extra testing etc.), then 41% of the programs which subsequently revealed potentially dangerous quality characteristics (which will be referred to as critical programs) would have been identified, but 27% of those selected would have been non-critical and would, therefore, have been given special treatment unnecessarily.

Thus, the quality indicator metrics only identify some of the critical programs, and they identify some programs as critical when they are not. However, the indicator metrics would have selected a group of 44 programs out of 226, and 32 (73%) of the programs selected would have proved to be critical programs. This result is significantly better than chance, since only 15 out of a random sample of 44 programs would be expected to be critical (based on the observed proportion of 79/226 = 0.35 critical programs).

Each indicator metric can be evaluated in terms of each quality characteristic metric separately as shown in Table 1. For example, this shows that of the 30 programs with large values of the information linkage metric, 7 programs also had a large number of changes, while 19 programs with a large information linkage value had a large value for at least one of the quality characteristics.

From Table 1, it can be seen that information linkage is a poorer indicator of critical programs than either control flow or size, since a larger proportion of programs with high values...
Table 1  Number of programs with large indicator metric values quality characteristic metric values

<table>
<thead>
<tr>
<th>Quality characteristic metrics</th>
<th>Indicator metrics</th>
<th>IL</th>
<th>CF</th>
<th>LHI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of programs</td>
<td></td>
<td>30</td>
<td>23</td>
<td>25</td>
</tr>
<tr>
<td>CHES</td>
<td>23</td>
<td>7</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>KER</td>
<td>41</td>
<td>9</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>OE</td>
<td>47</td>
<td>13</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>Programs with one or more large value</td>
<td></td>
<td>79</td>
<td>19</td>
<td>22</td>
</tr>
</tbody>
</table>

Figure 2 Size against Total Known Errors

Figure 4 Informational Fan-out against Total Known Errors

Figure 3 Control Flow against Total Known Errors

Figure 5 Informational Fan-in against Total Known Errors

Figure 6 Informational Fan-in against Size
of the information linkage metric were not critical, and a slightly smaller percentage of error-prone and complex programs were identified, although a similar number of change-prone programs were identified. The information linkage metric was, therefore, marginally less effective than code metrics in detecting critical programs (except with respect to change-prone programs), and considerably less efficient than code metrics since it incorrectly identified a larger proportion of non-critical programs. This approach to metric evaluation was developed by Kafura and Canning [3] who use the term "yield" to indicate efficiency, and "coverage" to indicate effectiveness.

This result is in direct contradiction to Henry and Kafura's work, and is less encouraging than the Kafura and Canning study [3], which indicated that information flow was at least as good a predictor of error-prone procedures as size, and a better predictor than McCabe's metric (a control flow metric related to the number of branches in a program [5]).

Critical program detection efficiency is, however, much better when programs with high values of any one of three metrics are identified as critical than the efficiency achieved by a single metric. This demonstrates that the different metrics detect different critical programs and implies that the different metrics should not be regarded as simple alternatives.

Investigation of the programs which each indicator metric identified as critical showed that the information linkage metric detected 7 programs which were not detected by either of the code metrics, and the code metrics detected 13 programs that the information linkage metric did not detect. However, the code metrics showed a substantial overlap, with 20 programs being detected by both metrics. It, therefore, appears that the code metrics are essentially equivalent, but there is a distinct difference between the code and design metrics.

The relationship between metrics is shown in more detail in figures 2 to 5, which show number of errors cleared in each program plotted against size, control flow, informational fan-out and modified informational fan-in, respectively. Similar patterns are found using number of changes or subjective complexity as the 'y' axis. (Please note that some points on the plots correspond to multiple points, but such points are not explicitly identified.)

These plots raise a number of important points:

1. Although there is evidence of an overall trend for large values of control flow and size metrics to coincide with large values of errors, changes and subjective complexity, the relationships are fairly weak.

2. There are no simple linear relationships between modified fan-in and any of the quality characteristic metrics. The scatter plots show that in general programs with very large fan-in values are infrequently changed and exhibited relatively few errors. This effect may be explained by the relationship between size and modified fan-in shown in figure 6. This indicates that in general, programs with large fan-in values were relatively small and large programs had relatively small fan-in values.

A large fan-in usually indicates a program which provides a frequently used function, and which would probably be invoked on a large number of paths during subsystem use, and which could therefore compromise the reliability of the system if it contained many residual errors. It is therefore a good strategy to keep such programs relatively small. The pattern observed in figure 5 can be interpreted as an indication of a good design strategy for maximising the benefits of reuse within the subsystem, while minimising the dangers.

It is, also, important to note that using a synthetic metric for fan-in complicates any interpretation of metric values. The program with the largest modified fan-in is infrequently called by many other programs, so the comments with respect to its criticality are valid. However, the program with the second largest modified fan-in, has acquired its value because it reads from a large number of the data structures, it is only called by one other program.

3. There is some evidence of relationships between informational fan-out and the quality characteristics, but they are also weak, and indeed are less consistent than the relationships between size and control flow metrics and the quality characteristic metrics. Since no simple relationship between modified fan-in and the quality characteristic metrics was found, this implies that the relationship between quality characteristics and the information linkage metric noted above is solely a result of the relationships between informational fan-out and the quality characteristic metrics. Thus, the synthetic information linkage metric obscured the underlying cause of certain relationships. This is another indication that synthetic metrics may complicate the process of interpreting metric values.

4. In general, it is more difficult to detect change-prone programs than it is to detect error-prone programs. This is an indication there are different underlying causes for change-proneness and error-proneness.
Error-proneness occurs because some components of a system may be more difficult to build than others, and/or may be built by people of different abilities, and/or may be built more carefully than others and/or may be subjected to different patterns of testing and use. Change-proneness occurs because some components of a system contain more important and/or more visible features than other. Programs which are likely to change are interface programs which change in response to changes in the environment, and programs which include essential features of the system and are likely to be enhanced when users require additional facilities or better performance.

There are, of course, related causes, for example a frequently changed program may lose its cohesion and become error-prone. However, the results of this study indicate that changes and errors should not be treated as synonymous for the purposes of project control or for the purposes of metric validation studies.

SUMMARY AND CONCLUSIONS

In this investigation the information flow metrics did not appear as useful as they have done in other studies. It proved difficult to extract the exact metrics defined by Henry and Kafura, the metrics actually used were not as good indicators of critical programs as code metrics, and the underlying causes of the metric values were sometimes difficult to determine because the metrics were synthetics. Each of these points is discussed, in more detail, below. Problems with metrics extraction occurred for two reasons. Firstly, the published descriptions of the metrics were ambiguous, and secondly some of the primitive counts that are used to construct the metrics were difficult to collect manually. Without good metrics definitions it is difficult to see how metrics can be properly validated, and without data collection and analysis tools it is difficult to see how they can be used in practical software production environments.

Although the information flow metrics were not as good at identifying critical programs as code metrics in this study, they are still potentially useful, since they are likely to be available earlier in the development life-cycle than branch counts or lines of code.

The results of this study suggest that it might be a cost-effective procedure to apply more stringent development procedures to programs with high fan-out values. Extra time spent on 13% of the programs would have been 63% efficient (since 60% of the programs identified warranted additional development time), but would have only been 24% effective (since only 24% of the programs which warranted additional development time would have been identified). Extra time spent on a randomly selected 13% of programs would have been 33% efficient and 33% effective.

In addition, if code metrics are used to supplement information flow metrics, procedures for assigning extra development time are better, being 73% efficient and 41% effective.

It would also seem to be appropriate both to ensure that programs with large fan-in values are kept small and simple, and to subject programs with a large fan-out value to immediate additional design effort, in order to ensure that there is not a missing level in the system structure.

It was observed that there are problems with the information flow metrics, which arise as a result of the synthetic nature of the metrics (i.e. the fact that they are obtained by combining the values of a number of other counts). The main problem is that the information flow metrics may conceal underlying effects and lead to incorrect diagnoses of the status either of the system as a whole, or of individual components. In addition, a major problem with the particular synthetics suggested by Henry and Kafura is the occurrence of procedures which are spuriously assigned a zero procedural complexity because their informational fan-in or fan-out is zero.

It would, therefore, seem preferable to use design metrics based on primitive counts rather than synthetics, unless it is very clear how the values obtained from the synthetics may be interpreted. This implies that we must accept the multi-dimensional nature of software metrics, and develop techniques for multivariate data analysis and presentation that are acceptable to software engineers and software managers as well as to statisticians.

REFERENCES


ACKNOWLEDGEMENTS

My thanks to Sharon Aird for providing the subjective complexity estimates, and to Lesley Pickard and Sue Linkman for their help with data analysis.

This research was supported by the UK Alvey Directorate, as part of the Software Reliability Modelling project.