42

Complexity Measures for System Development Models

E.H. Ferneley[1], D.A. Howcroft[2] and C.G. Davies

[1]Department of Computing, Manchester Metropolitan University, Manchester, M1 5GD, UK
[2] Department of Computation, UMIST, P.O. Box 88, Manchester, M60 1QD, UK

Abstract

This paper discusses mechanisms to measure the inherent complexity of the formative stages of the software development process. The techniques presented concentrate on two major specification and design modelling techniques, namely hierarchy and network diagrams, which may be found in many current structured development methods. The measurement of hierarchy models considers the level of nesting or *inherited* complexity of processing activity. The measurement of network models considers the implicit complexity of the information flows terminating at and emanating from processes.

Keyword Codes: D2.7, D2.8, D2.10
Keywords: Maintenance, metrics, design methods

1. INTRODUCTION

The advent of system development methods was hailed as the way forward to the production of quality, reliable software systems. However, throughout the past two decades it has become increasingly obvious that this has not always been the case. In fact there is now a major crisis facing the software development community. Increasing amounts of development resources are being devoted to the maintenance of existing systems. It has been estimated that by the mid 1990s nearly 90% of the system development resources will be consumed by the maintenance of existing systems [1]. A major area of concern, therefore, is the ability to control and monitor the development process, the assumption being that this will reduce the costs of quality software production. This raises a major issue, how does one monitor and control the development of quality software?

To this end there are many active research areas. One such avenue of research, suggests that the utilisation of complexity assessment techniques will aid in the reduction of the problem, by guiding the developers towards the reduction of the inherent complexity of systems under development. This, then is the subject of this paper.

The crisis in the software community can be summarised as the product of poor quality software [2, 3]. As a result, software quality quantitative estimation has become a major issue. Most studies have concentrated on the measurement of the later stages of the development lifecycle, such as the count of lines of code. However, it is also important to direct attention to the earlier stages of the software lifecycle since many of the software quality problems have been considered as originating in these early phases [4]. Incorrect decisions at this stage can result in the most costly of problems and have a major impact on software quality. Attempts to correct errors at the implementation stage is considerably more expensive than rectification of errors during the earlier stages of the lifecycle [5] and so measurement at the initial design stage enables an assessment of quality before committing to code. ·A key factor in the enhancement of quality,

M. Lee et al. (eds.), *Software Quality and Productivity*

in these early stages of the development cycle, is the ability to keep the complexity of the models as simple as possible. Thus the concept of complexity is vital to this study. *Complexity* can be defined as the degree of difficulty in analysis, design, testing and implementation of software [6]. Quantitative software metrics, when defined and used correctly, have been shown to be useful indicators of software complexity [7], and maintainability [8].

In many design methods the modelling of individual processing units precedes the modelling of the communication of these units with each other and the outside world. The modelling of processes is frequently concerned with the representation of the sequential time-ordering of individual process actions. Later, consideration is given to how these processes communicate with each other in terms of information flow. Consideration, will therefore be given first to the measurement of sequential time-ordered models and then how this measurement can be applied to network models.

2. THE MEASUREMENT OF HIERARCHY DIAGRAMS

A hierarchy diagram is a sequential time-ordered model of a processing unit (for an example refer to figure 1). This consists of four structural component types - iteration, selection, sequence and the atomic elementary. The diagram is always read from left to right, it may be considered as an inverted tree with the top most node being the root component of the tree, and the elementary components being the leaf nodes.

In order to evaluate the design complexity of hierarchical models it is necessary to consider each individual path through the diagram. To derive an overall complexity rating for hierarchy diagrams a measurement is obtained based upon the number of linearly independent paths through a process. This value has to take into account the complexity that occurs as we traverse *up* the hierarchical tree from the leaf component. This is known as the nesting level or *embedded complexity.*

The hierarchy metrics assign gradings to branches in a structure, these grades are drawn from the theories of *control flow metrics* [9, 10]. These metrics are concerned with the flow of control through the structure of a program. The control structure, thus defined, is modelled by the utilisation of directed graphs where the nodes correspond to program statements and an arc from one node to another represents a flow of control between statements. This measure is useful because, assuming that the graph is strongly connected, it indicates the number of *basic paths* (or linearly independent circuits) which, when combined, generate all possible paths throughout the program.

2.1 Hierarchy Complexity Metrics

The hierarchy metrics presented below have their foundation in the 'classical' complexity metric as defined by McCabe [11]. Empirical studies have suggested that there is a relationship between McCabe's cyclomatic complexity measure, which measures the *basic paths* through a given process, and the future reliability and maintainability of that process [7; 12]. The mechanism for the measurement of hierarchy diagrams is as follows:

- Because elementary components are the simplest component type they are assigned an initial design complexity grade of **1**. Elementary components have an implicit complexity as they represent sequential processing, this complexity must be considered.

- Sequential components within the body of the diagram are not graded, since they do not increase the processing complexity in terms of path structure. Sequential components are essential to the decomposition of a structure. Any such sequential decomposition requires the sub-division of the model into logical processing tasks.
- Selectable components are graded by applying principles derived from McCabe's cyclomatic complexity measure by considering the number of *basic paths*. The complexity grade for selections is determined by the number of selectable branches. Hence, when a selection is encountered the corresponding grade is the number of selectable possibilities. To illustrate, consider component *H* in figure 1, this has been awarded a rating of **3** as there are three distinct paths. Similarly, should there be four selectable children then the node would be given a complexity grade of **4**.

Iterative components are assigned a grade of **2**. This decision is based on the premise that when an iteration is encountered the number of *basic paths* is **2**, because the iteration may or may not be invoked. Given that an iteration has a potentially infinite number of paths due to the nature of the construct, a more meaningful measure is to count the number of distinct paths. Although distinct iteration loops are not counted, the weight of the component will be reflected in the *Hierarchy Complexity Totals* (see below).

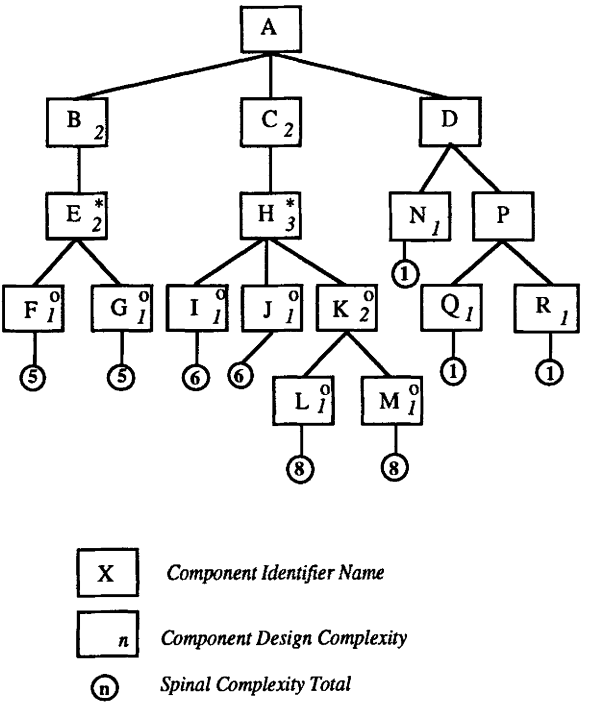


Figure 1: Example of Calculation of Spinal Complexity Total and Hierarchy Complexity Total

A grade for each *spine* in the hierarchy diagram is then calculated following examination of the spine from the root component to the specified elementary component, this is known as the *Spinal Complexity Total*. Such a traversal through the hierarchical diagram takes into account embedded design complexity. An overall grade for the diagram, the Hierarchy Complexity Total, may be attained by adding the constituent Spinal Complexity Totals as follows:

$$\text{Hierarchy Complexity Total} = \sum_{1}^{s} \text{Spinal Complexity Total} \qquad (1)$$

where Spinal Complexity Total is the individual spinal design complexity values and s is the number of individual spines. In the case of the diagram in figure 1 this would be 41.

The theoretical and practical basis of McCabe's metric has been heavily criticised [13, 14]. However, the application presented in this paper avoids the more contentious aspects of his work.

It considers embedded complexity that results from the nesting of structures, and also allows for distinct measurement of subsets of a program, rather than offering a single definitive value. Most metrics quantify the complexity of a process or program by means of a single global value. The use of a single-value number tends to conceal important internal aspects of the module, and thus degrades its usefulness. The metrics presented in this paper define a measurement technique that will integrate the embedded structural complexity of a process or program. This enables appreciation of every aspect of the hierarchy model and its contribution to complexity, rather than evaluating complexity as a whole.

3. THE MEASUREMENT OF NETWORK DIAGRAMS

A network diagram is a set of connected nodes, where the nodes represent processes (modules, functions or procedures, i.e. processing elements) and the connections represent the passage of data. To evaluate the complexity of network diagrams the connectivity of processes is examined. In order to achieve this the number of inter-process connections, along with their implicit structural complexity, is assessed. No distinction is made between local and global information flows, since complexity values are only based on an individual process' connectivity. The metric for individual processes therefore, reflects both the connectivity of the process and the implicit complexity of the structure of the data flowing into and out of that process.

3.1 Network Complexity Metrics

The network metrics presented here are loosely based on *information flow metrics* [15, 16] which measure the number of distinct information processing paths through a given process. To assess the complexity of process coupling, the total number of linearly independent paths through a process may be calculated. This is achieved by multiplying the number of information flows terminating at a process (known as *fan-in*) by the number of information flows emanating from a process (known as *fan-out*). Empirical studies have concluded that the level of information flow is closely correlated with development time [17]. These metrics are relevant to any system developed using a structured design technique which can be represented in terms of a network diagram (for example [18, 19, 20]).

The mechanism by which a total network complexity value for a given process node is achieved is as follows::

- Each information flow terminating at and emanating from the specified process is given a *Hierarchy Complexity Total* figure. This is based on the specified information flow's implicit data structure. For example, if a single parameter was being passed this would be **1**.

- To calculate a weighting for the overall complexity of information terminating at (or emanating from) the process the individual input (or output) *Hierarchy Complexity Total* figures are accumulated. The metric for the *total complexity* of information fanning-in to the process is therefore:

$$\textit{Fan-In Complexity Total} = \sum_{0}^{n} \textit{Hierarchy Complexity Total} \text{ terminating at the process} \qquad (2)$$

  Similarly, the metric for the *total complexity* of information fanning-out of the process is therefore:

$$\textit{Fan-Out Complexity Total} = \sum_{0}^{n} \textit{Hierarchy Complexity Total} \text{ emanating from the process} \qquad (3)$$

  Addition is used instead of multiplication to arrive at a value for the fan-ins (or fan-outs) of a node. The reasoning is best illustrated by example. Consider a single fan-in where the Hierarchy Complexity Total = 30; this could be perceived to have an equivalent complexity of three fan-in's with a Hierarchy Complexity Total of 10 each. If Multiplication was applied this would give a composite total of 1000, however, with addition this would give only a value

of 30. It may be considered that multiplication tends to inordinately distort the usefulness of the metric by severely skewing the total.

- The calculation for the composite figure for the module as a whole is weighted to reflect the difference between a single input (or output) source and multiple input (or output) sources. Therefore, in order to account for this, the *Fan-In Complexity Total* is multiplied by the total number of individual fan-in's giving a *Fan-In Multiplicity Complexity Total.* The *Fan-Out Multiplicity Complexity Total* metrics are derived in the same manner. This may be summarised as follows:

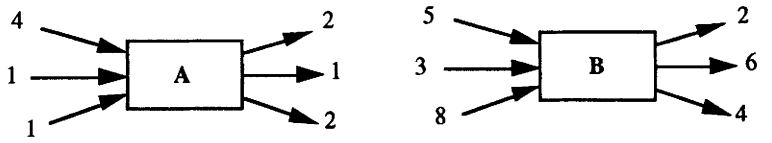
Fan-In Multiplicity Complexity Total = (*Fan-In Complexity Total* * ∑fan-in) (4)

Fan-Out Multiplicity Complexity Total = (*Fan-Out Complexity Total* * ∑fan-out) (5)

- Finally, to gain an assessment of the overall complexity for a given process the *Fan-In Multiplicity Complexity Total* is multiplied by the *Fan-Out Multiplicity Complexity Total.* Multiplication is used in deriving this value to reflect the total number of possible information processing paths through a given process. A problem would arise if either the number of fan-in's or fan-out's for a process is zero, since this would result in an instance of zero complexity. The solution proposed in such specialised cases is to assign a value of 1 whenever a zero fan-in or fan-out is encountered. The metric for arriving at a value for individual processes may be summarised as follows:

Process Complexity Total =
(*Fan-In Multiplicity Complexity Total* * *Fan-Out Multiplicity Complexity Total*) (6)

The metrics presented assess the complexity of the information entering and leaving the process and an overall information flow complexity measure is attained for each process. An example of the application of these metrics can be seen by considering figure 2.



A
Fan-in Multiplicity Complexity Total = (4 + 1 + 1) * 3 = 18
Fan-out Multiplicity Complexity Total = (2 + 1 + 2) * 3 = 15
Process Complexity Total = (18 * 15)

B
Fan-in Multiplicity Complexity Total = (5 + 3 + 8) * 3 = 48
Fan-out Multiplicity Complexity Total = (2 + 6 + 4) * 3 = 36
Process Complexity Total = (48 * 36)

Figure 2: Example of Calculation of Network Metrics

Traditional information flow metrics suffer from the inability to distinguish differing levels of complexity inherent within information flows. This deficiency would mean that the two processes illustrated in figure 2 would result in the same measure of complexity. Such traditional metrics would only distinguish three fan-in's and three fan-out's for each process. But this clearly is not so. It has already been shown in the discussion regarding hierarchy metrics that individual data

models necessarily have differing values of complexity. It follows therefore that the complexity of the individual information flows must also be variable. Any measurement of process and network complexity must take this into account. To this end, the network metrics presented here, permit a more meaningful measurement to be given to process and network complexity.

4. CONCLUSION

This paper has illustrated how two key complexity measurement techniques can be applied to two major forms of design specification. Enhanced quality may be brought about by attempting to maintain low levels of complexity. In order to achieve this the development process must be controlled. An essential part of this process is the ability to obtain meaningful measurements for complexity. This paper has attempted to show how the adaptation and modification of accepted metrics may be refined and thus applied.

The work presented in this paper has developed from a long-standing research project at UMIST into the construction of IPSE's and CASE tools. The latest CASE tool being developed is centred on the concept of method-configurable tools. This tool is being built incorporating the quantitative complexity measurement aids presented above. The research has shown, by the application of reverse engineering, a significant correlation between high values of complexity and error prone software. Consequently, the inclusion of such measurements is seen as an essential aid to the successful development of quality software products.

References

1. MOAD, J., Maintaining the Competetive Edge *Datamation*,, Feb. 1990, pp 60 -61.
2. FENTON, N. E., *Software Metrics: A Rigorous Approach*, Chapman and Hall, London, 1991.
3. SOMMERVILLE, I., *Software Engineering*, Addison-Wesley 4th Edition 1992.
4. HIRAYAMA, M., SATO, H., YAMADA, A. AND TSUDA, J. Practice of quality modelling and measurement on software life-cycle, *IEEE*, 1990.
5. BOEHM, B.W. *Software Engineering Economics* Prentice Hall, 1981.
6. FERNELEY, E. H., HOWCROFT, D. A. & DAVIES, C. G., Design Metrics as an Aid to Cost Estimation, Proc. ESCOM 1994, Ivrea, Italy, 11th - 13th May 1994.
7. CONTE, S. D., DUNSMORE, H. E. AND SHENN, Y. Y., *Software Engineering Metrics and Models*, Benjamin/Cummings, 1986.
8. KAFURA, D. & REDDY, R. R., The Use of Software Complexity Metrics in Software Maintenance in *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 3, Mar. 1987, pp. 335-343.
9. WILSON, R. I., *Introduction to Graph Theory*, Academic Press, 1972
10. LEGARD, H. & MARCOTTY, M., A Generalogy of Control Structure, *Communications of the ACM*, Vol. 18, Nov. 1975, pp. 629-639.
11. MCCABE, T. J., A Complexity Measure in *IEEE Transactions of Software Engineering*, Vol. SE-2, No. 4, 1976.
12. WARD, W. T., Software Defect Prevention using McCabe's Complexity Metric in *Hewlettt-Packard Journal*, Vol. 39, No. 3, 1989, pp. 30-35.
13. SHEPPERD, M., A Critique of Cyclomatic Complexity as a Software Metric in *Software Engineering Journal*, 3(2), 1988, pp. 30-36.
14. FENTON, N.E., Software Measurement and Analysis: A Case Study in Collaborative Research in *22nd International Conference on Systems Science*, IEEE, January 1989.
15. HENRY, S. & KAFURA, D., Software Structure Metrics Based on Information Flow in *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 5, 1981, pp. 510-518.
16. SHEPPERD, M., Early Life-cycle Metrics and Software Quality Models in *Information and Software Technology*, Vol. 32, No. 4, May 1990, pp. 311-316.
17. SHEPPERD, M., Design Metrics: An Empirical Analysis in *Software Engineering Journal*, 5(1), 1990, pp. 3-10.
18. MYERS, G., *Composite Structured Design*, Van Nostrand, 1978.
19. JACKSON, M., *System Development*, Prentice Hall, 1983.
20. YOURDON, E. AND CONSTANTINE, L.L., *Structured Design*, Prentice-Hall Inc., 1979.