



Design metrics for the Jackson method

E. Ferneley, D. Howcroft & C. Davies

*Department of Computation, UMIST, Manchester,
M60 1QD, UK*

ABSTRACT

The software design industry has still to reach maturity. Whilst later phases of the development cycle have established measuring techniques to assess a projects progress against, the application of early design metrics has yet to be fully embraced. The software industry has a potentially unique opportunity in that it is now becoming possible to technically assess the quality of the design prior to even prototype implementation.

The aim of this paper is to enhance the awareness of the initiators of software development projects by, firstly, highlighting existing metricing mechanisms that can assess the initial design phase of a software projects' development. The paper then presents a series of metrics that have been developed to measure the complexity of Jackson methodology specific designs.

The first set of design metrics presented assess the complexity of Jackson data structures. The metrics focus on measuring the complexity of control flow through the hierarchical structures. The second set of design metrics presented assess the complexity of Jackson network diagrams. The metrics examine the implicit and explicit connectivity complexity of modules or processes. Finally, limited implementation metrics are presented, specifically for examining the semantics of condition statements.

360 Software Quality Management

1. INTRODUCTION

The nature of software development forces critical decisions to be made during the initial design. Incorrect decisions, at this stage, can have a major impact on software quality, resulting in later software maintenance activity. The increasing complexity of software has been identified as a major factor in the exponential rise in software maintenance costs. At the initial design stage the developer has few means of validating design decisions and limited understanding of their consequences on the system. This paper presents a series of metrics that provide objective measurement of the complexity of graphical representations of software design.

The software industry has yet to reach maturity and hence the development of precise measuring techniques is still in its infancy. Metrics are quantitative measures of certain characteristics. Historically, much of the work in software metrics has concentrated on program code. However, it has been recognised that attempting to correct errors at the implementation stage is considerably more expensive than rectification of errors during the earlier stages of the lifecycle, Boehm¹. The earlier in the lifecycle that metrics are applied the greater the control of 'quality', Mills and Dyson². By the application of metrics at the software design phase, potential problem areas can be identified earlier, Troy and Zweben³. Measurement of design may be used to determine quality of design, allowing a comparison of different designs for the same requirement, Ramamoorthy et al⁴. It is hoped that the application of design metrics will aid the designers awareness of potential problems that may later arise in the lifecycle, providing a useful way of evaluating design quality before committing the design to code.

The purpose of the paper is to discuss the application of metrics to the measurement of models produced by the Jackson methodology. Currently no mechanisms exist for assessing the *complexity* of such diagrams. Complexity can be defined as the degree of difficulty in analysis, design, testing and implementation of software, Ramamoorthy et al⁴. The aim is not to derive a definitive measurement value for software complexity, rather to give an indication of the relative complexity of specific stages in the development of Jackson models. Therefore, the finest metric presented measures distinct subsets of modules or procedures. The higher level metrics only go as far as presenting value judgements on single modules or procedures. The intention is not to combine module metrics to give definitive values for specific pieces of software. It is the authors belief that unique values can not be assigned to something as complex as a software project.

2. THE PRINCIPLES OF JSP AND JSD

Jackson System Development (JSD) provides comprehensive coverage of much of the software development process, guiding the developer from the analysis of requirements through to the production of code, Jackson⁵. JSD is particularly suitable for larger systems involving a series of communicating sequential processes. It incorporates Jackson Structured Programming (JSP), Jackson⁶, which is one of the most widely used structured programming methods and is particularly suitable for data processing applications.

2.1 THE DEVELOPMENT STAGES OF JSP AND JSD

The basic principle of JSP is that programs are modelled on the structure of the data they are to process. Figures 1 & 2 provides the nomenclature of a Jackson diagram. The diagrams are read from left to right and based on the principles of sequence, selection and iteration.

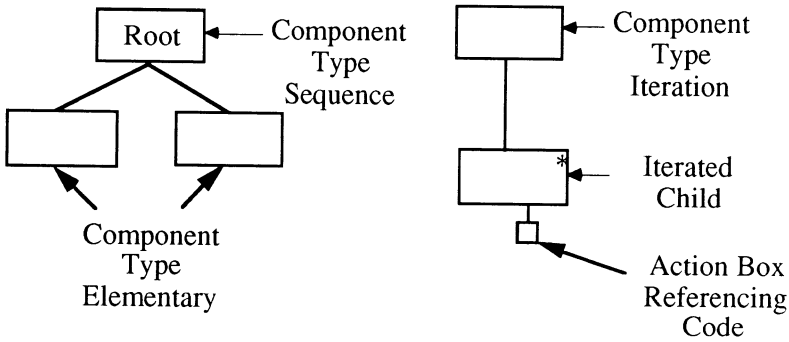


Figure 1: Sequence & Iteration nomenclature of Jackson Hierarchy Diagrams

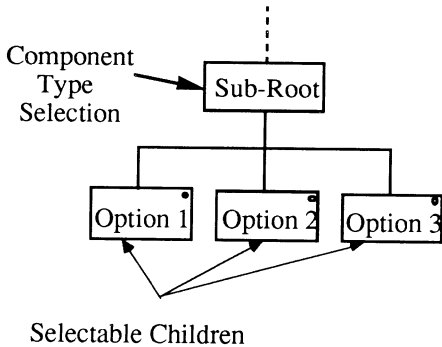


Figure 2: Selection nomenclature of Jackson Hierarchy Diagrams

362 Software Quality Management

Data models are constructed based upon the inputs and outputs of the program. A program structure is created by corresponding common nodes in the input and output structures, thus providing the basic model framework for the program. The actual program is constructed by allocation of program statements to appropriate parts of the model.

JSD is suitable for a more wide scale problem involving a set of communicating programs. The first stage of the method is the identification of individual actions and their associated entities. The individual JSD models (or model processes) represent the grouping of associated actions into hierarchical tree diagrams, using the same notation as derived for JSP. The resultant model processes show the time ordered sequence of actions that the entity will engage in throughout its life.

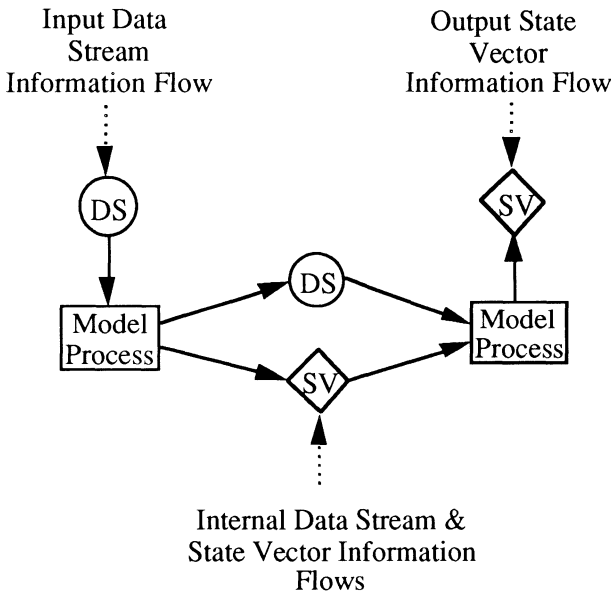


Figure 3: Example System Specification Diagram

Figure 3 shows a diagrammatic example of a network diagram, known as a System Specification Diagram (SSD) which is then derived. This shows the root components and connectivity of each of the model processes. The communication that exists between processes is represented by the passing of data, this data may be represented using JSP hierarchy diagrams.

3. WHY COMPLEXITY METRICS ?

A set of metrics are presented that measure the perceived complexity of software design and implementation details as seen by the system designer or programmer. This is distinct from computational complexity which refers to elements such as algorithmic efficiency.

Objective measurement has been chosen since subjective allocation of values relies on individual perception. In an heuristic approach to the measurement of software quality a set of 'characteristics' are used as a benchmark to assess the software. These characteristics must be present to a sufficient degree to constitute acceptable or 'quality' software. Such characteristics are also known as software quality factors, McCall et al⁷, Boehm et al⁸. A qualitative assessment of the degree to which these factors are present is common. However, ideally a series of structured or mathematical checklists which will allow the verification and measurement of the criteria and factors will have been established; these are known as software quality metrics. Such measurements or metrics allow an objective, quantitative assessment to be made.

Quantitative software metrics have been recognised as providing a good indication of program complexity and maintainability, Conte et al⁹, Kafura and Reddy¹⁰. By incorporating metrics at the design stage potential implementation complexity can be identified at the initial stages of the lifecycle, providing early, quantitative, project management feedback.

In order to objectively measure the complexity of Jackson models, derivations from McCabe's Cyclomatic Complexity¹¹ and Shepperd's Information Flow Metric¹² have been applied to assess design complexity. Halstead's Software Science¹³ has been used as a foundation for the measurement of implementation complexity details.

4. APPLICATION OF McCABE's CYCLOMATIC COMPLEXITY MEASURE TO JACKSON HIERARCHY DIAGRAMS

In order to evaluate the design complexity of Jackson hierarchy diagrams the complexity of each basic path (or spine) throughout the diagram is considered. To derive an overall complexity rating for a hierarchy design consideration is given to the embedded structure of each elementary component. This structure is based upon the relationships sequence, selection and iteration. A value is derived from the number of independent logical paths through a process. This value has to take into account *embedded* complexity, that is, the complexity that occurs as we traverse up the hierarchical tree from the position of the elementary component.

364 Software Quality Management

4.1 McCabe's CYCLOMATIC COMPLEXITY MEASURE

The hierarchy metrics applied have their foundation in McCabe's cyclomatic complexity number ($v(G)$) which measures the number of basic paths through a given procedure. The resultant cyclomatic complexity figure is said to have a bearing on the testability and maintainability of the program, Conte et al⁹, Henry and Selig¹⁴. McCabe associates a program with a directed graph which has unique entry and exit nodes. Each node in the graph represents a sequential block of code in the program, the flow of control between them is represented as edges (or lines connecting nodes). The complexity equation is defined as follows:

$$v(G) = E - N + 2$$

where E is the number of edges and N is the number of nodes.

4.2 JC-S AND JC-H DESIGN COMPLEXITY METRICS

M McCabe's complexity measure is used as a basis for assessing the complexity of control flows within Jackson hierarchy models. The four basic constructs of both JSP and JSD hierarchy models are elementary components, sequence, selection and iteration. Each of these elements have been assigned a different complexity rating. To aid in this discussion consider figure 4.

- Elementary components, given that they are the simplest component type are assigned an initial design complexity rating of 1.
- Sequential components within the body of a hierarchy diagram are not rated because they are modelled for clarity of design. They are simply a refinement of the parent and can be concatenated. With reference to elementary sequential components, it may also be argued that they are only distinct for clarity. However, it is assumed that any such sequential decomposition is due to sub-division into logical processing tasks. Therefore, distinct elementary component complexity ratings are given to reflect the sub-division.
- Selectable components are weighting by applying principles from McCabe's Cyclomatic Complexity Number; consideration is given to the number of linearly independent paths. The complexity rating for selections is determined by the number of branches. For example, when considering the component H ; in this case the component has been given a design complexity rating of 3 as there are 3 distinct paths. similarly, should there be four selectable children then the parent would be given a complexity rating of 4.
- Iterative components are assigned a rating of 2. This decision is based on the theory that an iteration has a potentially infinite number of paths due to the backward branching of the construct. Therefore, a



more meaningful measure is to count the number of distinct paths. This can, perhaps, most clearly be illustrated using the example of an iteration that is to be performed 100 times. If each distinct iteration was regarded as an independent path thorough the program, to award this a rating of 100 would necessarily skew the design complexity rating. Although distinct iterations are not counted, the scope of the component will be reflected in the final hierarchy figure as the iteration will be counted as many times as there are sub-branches to the specified component.

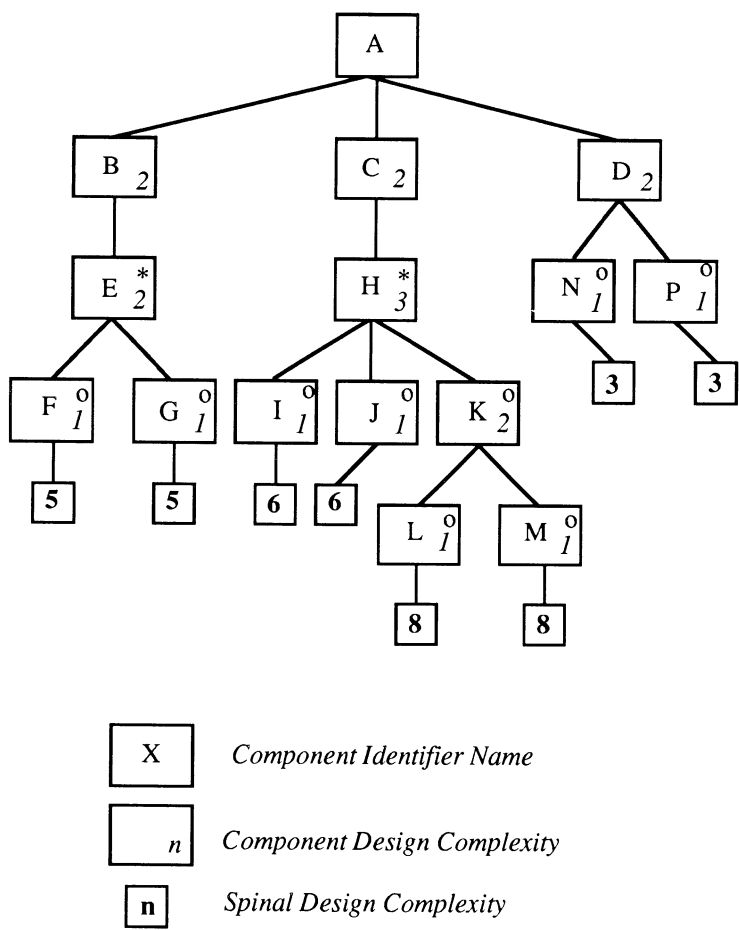


Figure 4: Example of Calculation of JC-S_n & JC-H



366 Software Quality Management

A design complexity figure for each spine in the hierarchy diagram is then derived (JC-S_n) by examining the spine from the root component to the specified elementary component. Such a traversal through the tree hierarchy takes into account embedded design complexity. An overall design complexity figure for the diagram (JC-H) may be attained by adding the constituent JC-S figures (In the case of the diagram in figure 4 this would be 44):

$$JC-H = \sum_{1}^s JC-S$$

where JC-S is the individual spinal design complexity values and s is the number of individual spines

4.3 CRITICISMS OF McCABE's CYCLOMATIC COMPLEXITY MEASURE

The Cyclomatic Complexity measure has been criticised as no more than a simple function of the number of decisions in a program, Fenton¹⁵. McCabe has also been criticised for failing to take full account of the program structure, the counting rules for control structures have been the subject of the debate, Shepperd¹⁶, Myers¹⁷. For example, nested loops are treated no differently than those in a sequence. Consider the following examples using nested IF's:

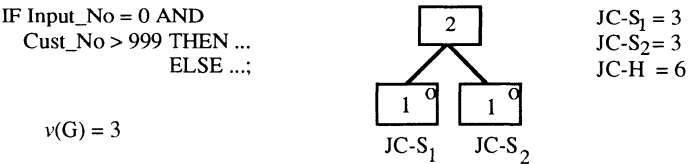


Figure 5a: Example A - Nested IF's

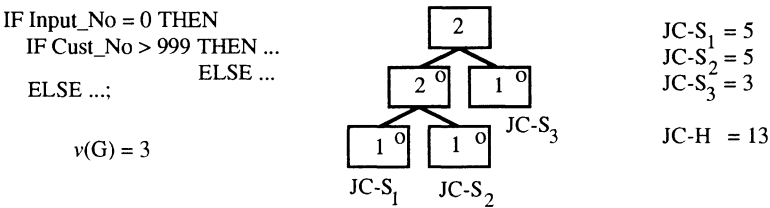


Figure 5b: Example B - Nested IF's

Examination of the two pieces of code gives an intuitive feeling that the second example is more complex than the first, yet this distinction is not reflected in the cyclomatic number. McCabe's application awards a

weighting which takes no consideration of the environment in which the decision resides. In the work presented here McCabe's philosophy is applied to the valuation of conditional elements of Jackson hierarchies. This foundation is then extended to derive overall spinal and hierarchical complexity figures, thus eliminating shortcomings as illustrated above.

5. APPLICATION OF SHEPPERD'S IF4 METRIC TO JACKSON NETWORK DIAGRAMS

To evaluate the design complexity of Jackson network diagrams the connectivity of model processes is examined. The number of (and implicit structural complexity of) the inter-process connections are assessed. Since complexity values are only to be assigned based on an individual processes connectivity no distinction is made between local and global information flows. The Jackson specific network metric (JC-N_n) for individual processes therefore reflects both the connectivity of the specified process and the implicit complexity of the structure of the data flowing into and out of that process.

5.1 SHEPPERD'S IF4 METRIC

The network metrics applied have their foundation in Shepperd's information flow metric (IF4) which measures the number of distinct paths through a given process¹². The IF4 metric is defined as follows:

$$\text{IF4} = (\text{fan-in} * \text{fan-out})^2$$

where fan-in and fan-out represent the number of information flows terminating at and emanating from a module.

The IF4 metric is distinct from the fan-in/fan-out metric as defined by Henry & Kafura¹⁸ in that no final 'line of code' (LOC) estimation is required; making the IF4 approach a more 'pure' design measurement. To assess the complexity of process coupling the total number of linearly independent paths is calculated by multiplying the fan-in figure by fan-out. The equation is squared to reflect the theory that understanding or modifying a component requires knowledge of both the component and the environment in which it operates.

5.2 JC-FIC, JC-FIC & JC-N NETWORK COMPLEXITY METRICS

The metrics presented assess the complexity of the information entering and leaving the process. An overall information flow complexity measure is then derived. To aid in this discussion consider figure 6.

368 Software Quality Management

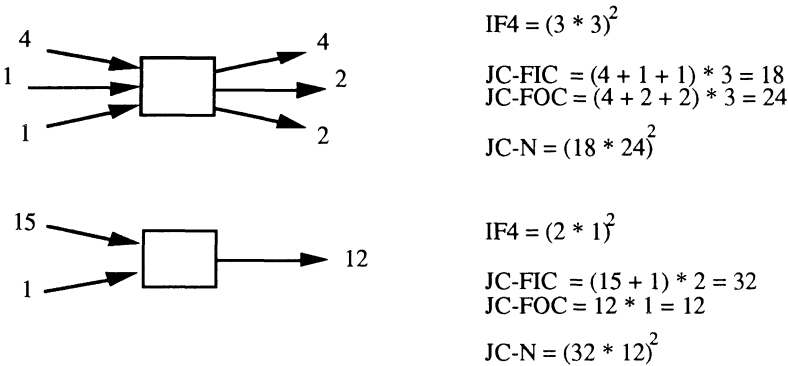


Figure 6: Example of Calculation of JC-FIC, JC-FOC & JC-N

The process by which a connectivity complexity figure is achieved is as follows:

- Each information flow terminating at and emanating from the specified process is given a JC-H figure. This is based on the specified information flows implicit data structure. This would, for example, be 1 if a single parameter was being passed.
- To calculate a weighting for the overall complexity of information terminating at (or emanating from) the process the individual input (or output) JC-H figures are accumulated. The metric for the total complexity of information flows terminating at (or fanning-into) the process is therefore:

$$JC-FI = \sum_{0}^n JC-H \text{ terminating at the module}$$

Similarly, the metric for the total complexity of information flows emanating from (or fanning-out of) the process is therefore:

$$JC-FO = \sum_{0}^n JC-H \text{ emanating from the module}$$

Addition was used as, for example, a single fan-in where JC-H = 30 was perceived to have an equivalent complexity to three fan-in's with a JC-H complexity rating of 10 each. Furthermore, the calculation for the composite figure for the specified network module (JC-N) is weighted to reflect the difference between a single input (or output) source and three distinct input (or output) sources.

- As is suggested in the previous section, to account for the number of input sources, the JC-FI figure is multiplied by the total number of fan-in's giving a composite fan-in number (JC-FIC). The composite

fan-out metric (JC-FOC) is derived in the same manner. The metrics for the complexity of information flows terminating at and emanating from a process, taking into consideration the implicit data structure complexity flow, are therefore:

$$\text{JC-FIC} = (\text{JC-FI} * \sum \text{fan-in})$$

$$\text{JC-FOC} = (\text{JC-FO} * \sum \text{fan-out})$$

Multiplication is used to reflect the total number of possible paths through a given process.

- Finally, to gain an assessment of the overall implicit and explicit information flow complexity for a given process the IF4 metric is then applied. JC-FIC is multiplied by JC-FOC and raised to the power of 2 to give an overall Jackson specific fan-in, fan-out complexity for the particular module. The formula is as follows:

$$\text{JC-N} = \sum (\text{JC-FIC} * \text{JC-FOC})^2$$

5.3 CRITICISMS OF SHEPPERD'S IF4 METRIC

Consider the case of zero fan-in or fan-out information flows; by strict application of IF4 the resultant JC-N would be zero. This would not be an accurate reflection of the processes connectivity complexity. Furthermore, if addition was used instead of multiplication the resultant equation would take no consideration of the number of independent paths through the given module. Therefore, in such a specialised case, the value of 1 is assigned to a zero fan-in or fan-out representation.

Additionally, IF4 only considers the number of connected modules and so ignores duplicate flows. As the complexity figure for the Jackson network diagram is concerned with the implicit data, underlying module connectivity duplicate flows are considered.

6. APPLICATION OF HALSTEAD'S SOFTWARE SCIENCE METRICS TO IMPLEMENTATION DETAILS ASSOCIATED WITH JACKSON DIAGRAMS

Halsteads philosophy, founded upon the identification of unique operators and operands, has been applied as a basis for the measurement of the complexity of implementation details in Jackson hierarchy models. More specifically the weighting mechanism of operators and operands has been applied to the conditions associated with branches in Jackson structures.

370 Software Quality Management

The objective of software science is to quantitatively measure programs using vocabulary measures, in particular operators and operands, since these are the basic units distinguishable by a compiler. Halstead uses these counts to derive nine software science indicators, the three most frequently referred to being token count (or vocabulary size), volume and effort. The metrics presented here focus on the token count aspect of the software science measure.

6.1 JC-DCI AND JC-I IMPLEMENTATION COMPLEXITY METRICS

Complexity measures for Jackson implementation details are based on vocabulary size and effort by the identification of unique operators and operands, as defined by Halstead¹³. For illustration, the metric as applied to the conditions of selections and iterations associated with Jackson hierarchy diagrams is first introduced. The metric may also be applied to implementation details associated with elementary components. In order to evaluate condition complexity, target language syntax must be taken into consideration. To aid in this discussion consider figure 7.

For the purposes of this paper Pascal has been used as the implementation language, a subset of the Pascal Counting Strategy as defined by Purdue University Software Metrics Research Group has been applied, Conte et al⁹.

The condition associated with the *Calc_Min_Wage* component: IF *Nett_Pay* < 90, has 1 unique operator (<) and 2 unique operands (*Nett_Pay*, 90). Therefore the complexity figure for the entire condition would be 3 (1+2). Halstead's application includes command names in the count. However, the WHILE has already been considered at the design level, therefore the complexity rating need not be incremented again.

It is feasible to calculate overall design and condition figures by combining the design and condition complexity numbers. For example, when considering figure 7, composite design and condition complexity figures can be assigned to each elementary component in the same way as composite design complexity figures are calculated for JC-S. The spine from the root component to the specified elementary component is examined. For example, in the case of *Calc_Min_Wage* the composite design and condition complexity figure is 13.

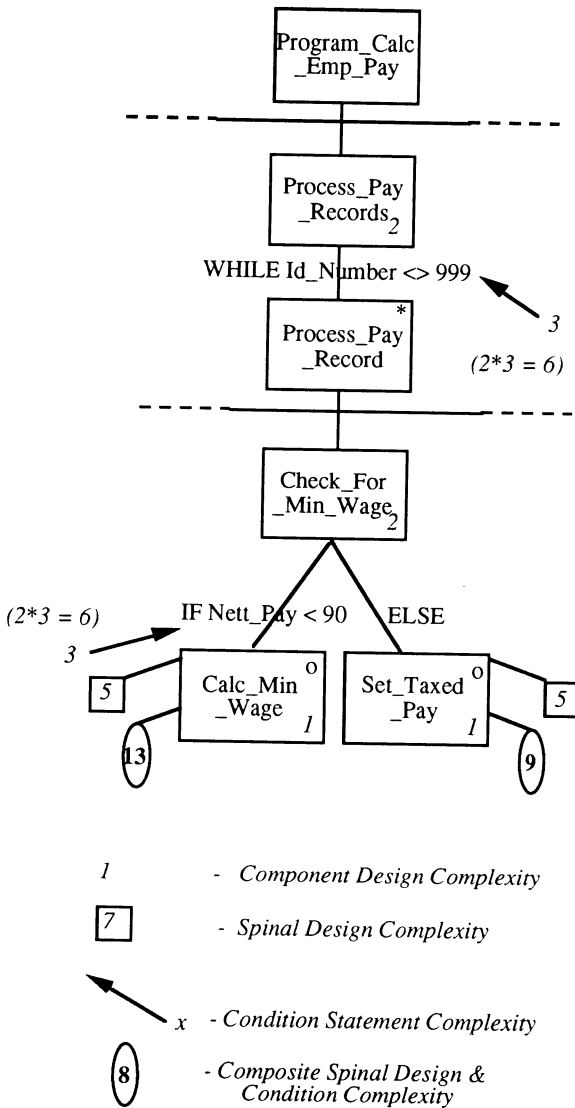


Figure 7: Example of Calculation of JC-DCI & JC-I

The figure for *Calc_Min_Wage* is derived as follows:

- Design and condition complexity figures associated with components responsible for control flow are multiplied together. In the case of *Check_For_Min_Wage* the composite design and condition

372 Software Quality Management

complexity figure would be $(2*3) = 6$. The 2 being based on the design complexity and the 3 from the condition: IF `Nett_Pay < 90`.

- To gain an overall design and condition complexity figure for the elementary component *Calc_Min_Wage* the embedded complexity must be considered. Therefore all figures associated with the spine from root to specified elementary component must be combined: $6 + 6 + 1 = 13$.
- To gain an overall design and condition complexity rating for the entire diagram the individual spinal design and condition complexity values (JC-SI), their representation being at each elementary component on the diagram, are added together:

$$JC-DCI = \sum_{i=1}^s (JC-SI_i)$$

where JC-DCI is the composite spinal design and condition complexity value and s is the number of individual spines

Finally, it is feasible to assign a complexity value to the implementation details associated with elementary components. As with condition complexity, a value is achieved by the identification of unique operators and operands. When implementation complexity numbers have been assigned to each elementary component, an overall complexity figure accounting for design, condition and implementation complexity may be achieved.

6.2 CRITICISMS OF HALSTEAD'S SOFTWARE SCIENCE METRICS

Halstead combined the results gained from his token count calculations with theories from cognitive psychology in order to predict the mental effort and time required to write programs. This application has been widely criticised in that theories from cognitive psychology have been inappropriately applied, Coulter¹⁹. The three main issues that are of concern are regarding the limitations of short-term memory, searches in human memory, and programming time. However, the application of Halstead's work presented here does not use these more contentious aspects.

7. DISCUSSION

Using information flow metrics for underlying measurement, it has been shown that there is a significant correlation between complexity and maintenance activity. High correlation was found between system change and complex procedures, Henry and Kafura¹⁸. Software maintenance expenditure now accounts for in excess of \$30 billion per

annum, Rock-Evans and Hale²⁰. Therefore, identification of potential complexity in the early phases of software development is essential. Such early identification and elimination of software complexity should therefore significantly reduce the time, effort and money consumed during the maintenance phase of the software development lifecycle.

A set of complexity metrics have been presented covering the design, condition and implementation stages of Jackson models. The intention being to provide design and implementation support for software developers. To collate and calculate the metrics presented is an extremely time consuming task. Therefore, an automated metricing tool is being incorporated into a Jackson CASE tool, using X windows and Motif, that is under development at UMIST. In addition to automatically generating JC metrics valuations the tool will also provide an objective LOC count facility. Although the focus has been on Jackson models the theories presented are equally applicable to other graphical design methods. The major strength of the theories presented is that implicit as well as explicit connectivity is assessed.

REFERENCES

1. Boehm, B.W. *Software Engineering Economics* Prentice Hall, 1981.
2. Mills, H.D. and P. B. Dyson. 'Using Metrics to Quantify Development.' *IEEE Software*, 7, 2:15-16, 1990
3. Troy, D.A. and S. H. Zweben. 'Measuring the Quality of Structured Design' *Journal of Systems and Software* 2:113-120, 1981
4. Ramamoorthy, C.V., Tsai, W., Yamaura, T. and Bhide, A. 'Metrics Guided Methodology' in *COMPSAC '85: Computer Software and Applications Conference*, October 1985, Chicago, IEEE.
5. Jackson, M. *System Development*, Prentice Hall, 1983.
6. Jackson, M. A. *Principles of Program Design*, Academic Press 1975
7. McCall, J. A., Richards, P. K. & Walters, G. F. *Concepts and Definitions of Software Quality: Factors of Software Quality, Vol. 1* NTIS, 1977
8. Boehm, B. W., Brown, J. R., Kaspar, H. et al. 'Characteristics for Software Quality' *TRW Series of Software Technology, Vol. 1* Amsterdam, 1978
9. Conte, S. D., Dunsmore, H. E. and Shenn, Y. Y. *Software Engineering Metrics and Models*. Benjamin/Cummings, 1986
10. Kafura, D. and Reddy, R. R. 'The Use of Software Complexity Metrics' *IEEE Transactions on Software Engineering*, SE-13,3:335-343, March 1987
11. McCabe, T. J. 'A Complexity Measure', *IEEE Transactions of Software Engineering*, SE-2,4, 1976
12. Shepperd, M. 'Early life-cycle metrics and software quality models' *Information and Software Technology*, 32,4:311-316, May 1990.



374 Software Quality Management

13. Halstead, M.H. *Elements of Software Science*, New York, Elsevier, North-Holland, 1977.
14. Henry, S. and Selig C. 'Predicting Source-code Complexity at the Design Stage' *IEEE Software*, 36-44. March 1990.
15. Fenton, N.E. 'Software Measurement and Analysis: A Case Study in Collaborative Research' *22nd International Conference on Systems Science*, Pubs IEEE, January 1989.
16. Shepperd, M. 'A critique of cyclomatic complexity as a software metric' *Software Engineering Journal*, 30-36, March 1988.
17. Myers, G.J. 'An extension to the cyclomatic measure of program complexity', *SIGPLAN Notices*, 12,10:61-64, 1977.
18. Henry, S. and Kafura D. 'Software Structure Metrics Based on Information Flow' *IEEE Transactions on Software Engineering*, 510-518, September 1981.
19. Coulter, N.S. 'Software Science and Cognitive Psychology', *IEEE Transactions on Software Engineering*. SE-9,2:166-171. March 1983.
20. Rock-Evans, R. and Hales K. 'Reverse Engineering: Markets, Methods & Tools' *Ovum Ltd*, 7 Rathbone Street, London W1P 1AF, 1990