Failure & Lessons Learned in Information Technology Management
Vol. 2, pp. 83–90, 1998

1088-128X/98 $10.00 + .00

Halstead's Metrics Revisited for Knowledge-Based Systems

H. R. MORENO AND R. T. PLANT

Department of Computer Information Systems, University of Miami, Coral Gables, FL 33124

As organizations become more virtual in nature and utilize knowledge-based technologies to support this mode of operation, the requirement to create robust industrial-strength systems becomes vital. The project managers and systems developers require tools and metrics to ensure effective planning and monitoring of the systems development. The article focuses upon Halstead's software science effort metric. The article shows the weakness and inapplicability of this method when applied to knowledge-based systems development and indicates the potential hazards associated with the incorrect selection of metrics.

Software metrics; Knowledge-based systems; Software estimation

INTRODUCTION

The organization of the 21st century is destined to be significantly different from that of the 20th century. The primary cause is the need to be flexible, agile, and adaptable in a marketplace that demands mass customization of products. This move to a virtual organization (Davidow & Malone, 1992), in which the organization is flexible both internally through its ability to reconfigure its processes and externally through its connectivity with customers, suppliers, outsourcers, etc., is based upon network technologies. The intranets that connect the internal processes and the internets that link the organization to the external value system are key to this new organizational structure. However, the telecommunications component is just one aspect of the virtual organization. It is the ability of that organization to process information and have its systems autonomously behave in a knowledgeable manner upon that information that provides the competitive advantage and virtualization to those organizations. The autonomous intelligence is facilitated by the use of artificial intelligence in the form of agents, facilitators, and knowledge query and manipulation languages. As stated by O'Leary, Kukka, and Plant (1997), "virtual organizations may be the first large-scale industrial application of artificial intelligence." A strong component of this is the use of industrial-strength knowledge-based systems (KBSs), which perform a variety of tasks in the Intelligent Enterprise (Quinn, 1992). KBSs work in areas such as marketing, acting as interpretive agents of corporate data mining, call center management in which the operator interacts with a KBS to advise customers (Qunin, 1992), and finance where KBSs are used for capital investment appraisal (King & McAulay, 1995). The KBS component of AI is perhaps the most mature aspect of AI and as such the most frequent to be integrated into a systems development or embedded into other systems.

The movement, and hence increased operational dependence of organizations on KBSs, requires the organization understand the technology in a context other than that of customized research-based systems. This requires an understanding of the interaction of the technology with the culture, structure, process, and innovation of the organization as well as the implementation of industrial-strength KBS from a project management perspective.

A critical aspect of all project management and software development is the ability to understand the relationship between design and factors, such as: reliability, quality, complexity, maintainability, and structure. In order to understand these relationships it is necessary for the developer to appraise many interrelated parameters, such as the way a software language is used, modularity within design, and the complexity of a design. To study these aspects of development a branch of software engineering has developed, termed Software Metrics, in which the theory of measurement is applied to software systems. The use of metrics in the development and management of traditional systems is well documented (Adamov & Baumann, 1987; Fenton, 1991; Zuse, 1991). However, even though there is a growing literature on the management of systems development through metrics (Garmus & Herron, 1996; Grady, 1992; Grady & Caswell, 1987) and of knowledge-based systems development and project management (DeSalvo & Liebowitz, 1990; Liebowitz, 1991; Turban & Liebowtiz, 1992), the literature relating to metrics and KBSs is sparse, primarily relating to issues in their validation and verification (SAIC, 1995). The aim of this article is to perform an introductory investigation into the issues surrounding the construction and application of metrics to KBSs. The

Correspondence and requests for reprints should be addressed to R. T. Plant. Tel: (305) 284-1963; Fax: (305) 284-5161; E-mail: rplant@exchange.sba.miami.edu.

article is organized as follows: the second section gives an overview of three types of software metrics, structural, data, and token based, that can be used to identify the properties of systems. The third section focuses upon one of these metrics, Halstead's token-based metric for measuring the development effort required by a software system in its construction. An experiment is performed to examine the applicability of this metric in order to determine the effort required in the creation of a KBS and what variances occurred as the tokens used in the metric were changed in the measurement process. The fourth section reflects upon the success of this experiment and the use of metrics to determine development effort for KBSs. Finally, the article discusses the future use of metrics in relation to KBS development.

METRICS OVERVIEW

The area today known as Software Metrics has its origins in the mid-1970s when two highly influential works were published: McCabe's (1976) "A Complexity Measure" and Halstead's (1975) *Elements of Software Science*. Since then there has been continuing research in the area of metrics (for a full reference see Conte, Dunsmore, & Shen, 1986; Fenton, 1991) that primarily falls into three categories: logic structure metrics, data structure metrics, and token measurement metrics.

Structural Metrics

The logic structure metrics are concerned with identifying and counting the decisions and branching factors of a program (Zuse, 1981). The flow of control within a program is examined and the occurrence of forward, backward, or horizontal branches is analyzed, taking into account whether they are compound conditions or how deeply nested are the predicates. An example of this type of metric is that of McCabe's Cyclomatic Complexity Number [$v(G)$] (McCabe, 1976), the value of which is determined by the equation:

$$v(G) = e - n + p$$

where e and n are the number of edges and nodes in a flowgraph and p is the number of connected components.

The cyclomatic complexity metric in conjunction with other logic structure metrics, such as Zolnowski's depth of nesting metric (Zolnowski & Simmons, 1981) and the Schneidewind-Hoffmann reachability metric (Schneidewind & Hoffman, 1979), aim to allow software engineers to reason about the complexity of a programs flow of control, such that the lower the complexity the easier a system is to maintain, test, and reason over.

Data Structure Metrics

The second set of metrics is the data structure metrics. These are metrics associated with measuring information flow characteristics. They can be at the program level, intramodule level, or variable level. A series of metrics that attempts to measure information flow between modules has been proposed by Henry and Kafura (1981) (subsequently modified by Sheppard, 1990). The Henry–Kafura information flow complexity metric:

$$\text{program-length} * (\text{fan-in} * \text{fan-out})^2$$

utilizes measurements of *fan-in* and *fan-out*, which Fenton defines as the "fan-in of a module M is the number of local flows that terminate at M, plus the number of data structures from which information is retrieved by M" and the "fan-out of a module M is the number of local flows that emanate from M, plus the number of data structures that are updated by M" (Fenton, 1991). Further to these information flow metrics, the software engineer can utilize measures such as *coupling* or *cohesion*, where coupling can be defined as "a measure of the degree of interdependence between modules" (Pressman, 1987) and determined through Fenton's coupling metric (Fenton, 1991), which is defined as:

$$c(x,y) = i + (n/n + 1)$$

where the coupling c between modules x and y is related to n, the number of interconnections between those modules, and the coupling type, i, being a numeric value for the worst case coupling between x and y. The coupling types being on a scale from contents (5), common (4), control (3), stamp (2), data (1), no coupling (0). Cohesion is similarly defined by Fenton: "Cohesion is an attribute of individual modules, describing their relative functional strength i.e., the extent to which the individual module components are needed to perform the same task" or in numeric terms as:

> Cohesion Ratio = number of modules having functional cohesion/total number of modules.

The data structure metrics are important to the software engineer as they can be examined early in the life cycle and development of a system, acting as cost-saving measures. They also function as tests that ensure the system is being developed in accordance to a given methodology (i.e., a balance between low coupling and high cohesion), and poor metric results would indicate weakness in the development techniques being used.

Token-Based Metrics

The third set of metrics is those based upon measurement of tokens in a system, program, or module. This

approach to measurement was initiated through Halstead's Software Science, where he proposed a series of metrics based upon the measurement of operands and operators in an algorithm. The basic metrics are defined as:

n_1 = number of unique operators
n_2 = number of unique operands
N_1 = total usage of all operators
N_2 = total usage of all operands

from which Halstead defined the vocabulary n as:

$$n = n_1 + n_2$$

the implementation length N as:

$$N = N_1 + N_2$$

and the volume V as:

$$V = N * \log_2(n).$$

Halstead developed a whole series of more complex metrics based upon these basic token measures that measure such characteristics as Effort, Program Level, and Level of Program Abstraction.

The work of Halstead and the Software Science movement has not been without its controversy and criticism (Hamer & Frewin, 1982; Shen, Conte et al., 1986). However, the majority of this has been aimed at those metrics that are based less on direct token measurements and more on those metrics that draw from a theory of psychological measurement. An example of such a metric is the Intelligent Content metric I, where:

$$I = L\hat{} * V$$

$L\hat{}$ being the estimated level of abstraction of the implemented algorithm:

$$L\hat{} = 2/n_1 * n_2/N_2.$$

The use of psychological techniques such as the Stroud-Number (Stroud, 1967) led to significant criticism and the metrics relating to development time estimation and the prediction of bug-rates can largely be disregarded. However, this should not distract from the value of the basic Software Science, token-based metrics, which are still valuable to the software engineer in estimating relative development effort. The Effort metric:

$$E = V/L\hat{}$$

can be used to provide a base for predicting the development effort of projects based upon experience and similar existing projects in similar development environments.

HALSTEAD'S EFFORT METRIC FOR KBS

Introduction

The previous section outlined three general categories of metrics that can be applied to software systems to extract a desired understanding of that systems properties. The categorization is obviously not complete, and full reference material is available on this subject (Adamov & Baumann, 1987; Fenton, 1991). A set of metrics within this field has been proposed and used to better determine the cost and effort required to build a software system. These include Function Points (Albrecht & Gaffney, 1983) and COCOMO, which is based upon the estimation of source lines of code in a program (Boehm, 1981). Metrics have also been used to help determine maintenance efforts (Harrison, Magel, Kluczny, & DeKock, 1982; Kafura & Reddy, 1987). However, as stated earlier, very little work has been performed on the development of metrics for assessing the development efforts required for KBSs. In the remainder of this article we will consider Halstead's token-based Effort metric in relation to KBSs. The article will examine the metric in detail and through experimentation examine what are the critical factors affecting this measure. We also consider what are the key influences of those factors, such that the software development/maintenance effort can be reduced.

Experimental Background

The aim of the experiment was to take Halstead's Effort metric and apply it to KBSs. In doing so two issues were to be examined: first, by manipulation of the parameters that comprise the Effort metric, establish an initial understanding of KBS design optimization (e.g., in KBS design does the number of operands or operators have the most significant effect upon the Effort required to develop the system) and second, to understand if the metric was of real industrial-strength use to establishing effort requirements for KBS development. These two issues will establish the utility of such a metric for knowledge engineers in the field.

Experimental Details

Halstead defined his Software Science to be based upon the measurable properties of algorithms and as such KBSs can be clearly measured in this way.

Knowledge-Based Systems

In this article we are using the term KBS to be a traditional production system. Such a system is configured in three parts: a rule base, which acts as the knowledge representation; an inference engine, which manipulates the rules; and a working memory, which acts as a store for intermediate results prior to determining the solution. We will now examine these in more detail.

Rule Base. A rule base consists of a series of production rules that are composed of a proposition and a consequence:

Rule1: IF Condition_1 THEN Action_1
Rule2: IF Condition_2 THEN Action_2
Rule3: IF Condition_3 THEN Action_3

Inference Engine. The rules of the knowledge base are manipulated by an inference engine, which acts in a three part cycle: Match, Select, Fire.

The inference engine first Matches the initial user input with the Condition sides of the rules, generally known as the LHS of the rule. When a match is found then the inference engine notes the match and progresses through all the rules until all matches are noted. The inference engine then performs the second part of the cycle, the Selection of the most appropriate rule. This is easy if there is only one match; however, this is rarely the case and thus a technique known as Conflict Resolution is utilized. There are many Conflict Resolution strategies, for example: select the first rule encountered, the last rule, the shortest, the rule with the most number of subconditions in the LHS, etc. Once a rule has been selected the inference engine then Fires the rule, in that the contents of the right-hand side of the rule are added to the Working Memory Environment (WME), the third major component of the KBS.

Working Memory Environment. The WME is the temporary storage area for the inference engine. Having performed the first Match, Select, Fire cycle the inference engine in subsequent cycles takes the contents of the WME and compares it to the LHS of the rules, acting upon this improved information input set. This continues until all possible rules have been fired for the given input WME and intermediate WMEs, at which time a result is produced and the cycle terminates.

Forward and Backward Chaining. The basic Match, Select, Fire cycle can occur in two ways: forward chaining and backward chaining through the rule set. Forward chaining utilizes a set of events and attempts to draw conclusions from them, deduction going from the LHS to the RHS of the rule, whereas backward chaining starts from the goal or expectation of what is happening and attempts to find a set of conditions that once satisfied will support that goal, deduction going from the RHS of a rule through the LHS.

VP-Expert. The tool that we utilize in this article, to develop the KBS models, is that of VP-Expert. This particular shell uses backward chaining and divides a program into three aspects: the action block, the rules block, and the statements block. The action block controls the inference process and determines the search criteria, the rules block contains the knowledge base for the system, and the statements block controls the system–user interaction. An example program is given in Figure 1. The declarative knowledge-based structure of VP-Expert is advantageous for experimentation as it aids standardization and consistency across models and their execution. Even though VP-Expert (Pigford & Baur, 1990) was the language used in this experiment the approach used is generic to any language in which the substantial number of industrial KBS run (e.g., CLIPS [NASA, 1991] or OPS5 [Forgy, 1981]).

In order to examine the effect that modifications to the system had upon the Software Science Metrics, it was necessary to construct a parser that automatically determined the values of the metrics from the programmed KBS. This parser was written in Clipper.

Constraints

In this preliminary investigation, only KBSs written in VP-Expert were examined with respect to Halstead's metrics, and the following metrics were computed: Total usage of operators, Number of unique operators, Total usage of operands, Number of unique operands, (Program) Length, (Program) Vocabulary, (Program) Implementation length, Purity ratio, Volume, Estimated error, Estimated level of abstraction, Effort, Cyclomatic complexity, Extended cyclomatic complexity, Total number of rules, Total number of premises. In order calculate these metrics several language constraints had to be identified and standardized. First, the aim of the experiment is primarily to focus the metric count upon the declarative aspects of the system and not consider the procedural (Action Block) component, and thus the semicolon operator could be ignored as the each of the blocks including the action block terminates with a semicolon. To overcome this missing operator in the rules block the THEN operator was included in the count, when normally only the IF operator would be counted. Examining the syntax of the Rule statement:

RULE<rule_label>
IF
<condition1> [AND/OR]
[condition2 [AND/OR]]
[etc.]
THEN
<conclusion1> [CNF N]
[conclusion2 [CNF N]]
[etc.]
[clause1...]
[clause2...]
[ELSE
<conclusion3> [CNF N]
[conclusion4 [CNF N]]
[etc]
[clause 1]
[clause 2]
[etc.]]
[BECAUSE <text>]

We see that the operators need to be constrained in order to clarify the experiment's results. Hence, several operators were not considered, such as BECAUSE, CNF. These operators were not utilized in the test programs upon which the experiment was based; however, their absence does not unduly influence the outcome. If

```
! Program Name:  Haus_All.KBS
!Action Block

ACTIONS
  FIND Potential   !Identifies the RHS of the rule (event) to satisfy through LHSs
  DISPLAY "The potential of the home is {Potential}~";   !Displays output

!Rules Block
RULE 01
IF  location = N_East              !LHS Condition 1
  AND  Num_Floors = 1              !LHS Condition 2
  AND  Num_Rooms = 1               !LHS Condition 3
  AND  Num_Baths = 1               !LHS Condition 4
THEN   Potential = aa;             !RHS Action

RULE 02
IF  location = N_East              !LHS Condition 1
  AND  Num_Floors = 1              !LHS Condition 2
  AND  Num_Rooms = 1               !LHS Condition 3
  AND  Num_Baths = 2               !LHS Condition 5
THEN   Potential = ab;             !RHS Action
.
.
.

RULE 256
IF  location = N_West              !LHS Condition
  AND  Num_Floors = 4              !LHS Condition
  AND  Num_Rooms = 4               !LHS Condition
  AND  Num_Baths = 4               !LHS Condition
THEN   Potential = jy;             !RHS Action

!Statements Block
ASK location: "Where is the location?";
CHOICES location: N_East, S_East, S_West, N_West;

ASK Num_Floors: "How many floors does the house have?";
CHOICES Num_Floors: 1, 2, 3, 4 ;
```

Figure 1. Example rule-based program.

they are added later then their effect will only be to increase the values of the metrics including that for the Effort metric. The entities shown in Figure 2 were counted as single operators.

Results

The Effort equation, as derived by Halstead, is a function composed of four independent variables and this can be stated as $E = f(N_1, N_2, n_1, n_2)$. In our initial investigation of the effects of the independent variables on the Effort, two knowledge bases were written (House1.kbs and House2.kbs). These two were identical in all respects with one major exception, the number of unique operands, n_2, in that House1.kbs has 256 and House2.kbs has 17. Table 1 gives a detailed accounting of the each of the knowledge bases.

The knowledge-based system House2.KBS contained 256 rules (the 256 rules result from there being four conditions in each rule, each condition having four possible values) and only four unique conclusions. These four conclusions were represented by four unique operands. (Halstead defines Operands as variables or constants that the implementation employs and Operators as

```
= < > <> >= =< IF THEN ELSE FIND CHAIN ASK AND OR
```

Figure 2. Operators.

Table 1. Software Science Metric Results

Metric	Knowledge Base	
	House1.kbs	House2.kbs
Total usage of operators (N_1)	2,565	2,565
Number of unique operators (n_1)	6	6
Total usage of operands (N_2)	2,565	2,565
Number of unique operands (n_2)	269	17
N (length) = $N_1 + N_2$	5,130	5,130
n (vocab) = $n_1 + n_2$	275	23
$N\hat{} = n * \text{LOG}_2 (n_1) + n * \text{LOG}_2 (n_2)$	2,186.733	84.997
Purity ratio = $N\hat{}/N$	0.426	0.017
V (volume) = length * LOG_2 (vocab)	41,569.866	23,205.873
$B\hat{}$(est. errors) = volume/$E0$ ($E0 = 3{,}100$)	13.410	7.486
$L\hat{}$ (est. level of abstraction) = $2/n_1 * n_2/N_2$	0.035	0.002
E (effort) = volume/$L\hat{}$	1,189,145.436	10,504,070.086
Cyclomatic complexity = number of decision statements + 1	257	257
Extended cyclomatic complexity = sum of decisions, ANDs, Ors + 1	1,025	1,025
Total numner of rules	256	256
Total number of premises	1,024	1,024

symbols or combinations of symbols that affect the value or ordering of Operands.) The operands in this experiment are the values of the condition identifier <condition>, and the conclusion identifier <conclusion>, as defined in the syntax of the VP-Expert rule structures given above. In declarative languages these are fixed identifiers, not variables as in procedural languages, and hence the declarative language requires a unique identifier for each unique item.

As can be seen from Table 1, the Effort has a value of about 10.5 million elementary mental discriminations. This KBS was modified to have 256 unique conclusions and renamed to House1.KBS. These conclusions were represented by 256 unique operands. As can be seen from Table 1, the computed Effort now had a value of about 1.2 million, whereas the cyclomatic values remained constant. The fact that the cyclomatic values remained constant was expected because this change in operands did not change any of the independent variables in the complexity equations.

The fact that the metrics show that effort decreases as the number of unique operands increase appeared to be a significant weakness in Halstead's metrics when applied to a KBS created in a declarative language. In order to examine this further it was decided to take the partial derivative of Effort with respect to each of the four independent variables. This was performed to show the rate of change in E in relation to each of the variables.

Figure 3 displays the Effort equation and the partial derivatives for each of the independent variables. Calculations were then made using values in the range of the House1.KBS and Hous2.KBS.

The actual values of the Effort equations are presented in Table 2.

The effect of each metric on the Effort equation is summarized in the following:

- As the number of unique operators increases the rate of change of effort increases linearly.
- As the number of unique operators increases the effort also increases linearly.
- As the total number of operands increases the Effort also increases linearly.
- As the total number of operands increases the rate of change of effort also increases.
- As the total number of operators increases the rate of change of effort remains constant.
- As the total number of operators increases the effort increases linearly.
- As the number of unique operands increases the rate of change of effort decrease assomtopically.

Observations on the Effort Metric

Metric N_1 is unusual in its effect on the Effort equation. It, and it alone, gives a constant partial derivative. The actual value of the partial derivative will vary if the initial values of the other three metrics are changed. However, once initialized, the rate of change of Effort due to a change in N_1 is constant.

The remaining three metrics, N_2, n_1, and n_2, share a common effect on Effort. They all have their maximum influences when their initial values are small and these effects diminish as their values increase.

Both metrics N_2 and n_1 increase the effort in a nonuniform manner with metric N_1 increasing the effort at a constant rate. This is in keeping with the general observations that increasing the quantity and usage of operators and operands increases the complexity of the algorithm.

Upon reflection, the effect of metric n_2 is not surpris-

N_1 = Total operator usage
N_2 = Total operands usage
n_1 = Unique operator count
n_2 = Unique operand count

E (effort) = K0*K1*K2

$K0 = N_1 + N_2$
$K1 = LOG_2(n_1 + n_2)$
$K2 = (n_1 * N_2) / 2*n_2)$

$E = (K0)*(LOG_2(n_1+n_2))*(n_1*N_2/2*n_2)$

$\delta E/\delta n_1 = (K0*K2)/(n_1+n_2)*(LN(2)) + (K0*LOG_2(n_1+n_2))*N_2/(2*n_2)$
$\delta E/\delta n_2 = (K0*K2)/(n_1+n_2)*(LN(2)) - (K0*K2*LOG_2(n_1+n_2))/n_2$
$\delta E/\delta N_1 = K1*K2$
$\delta E/\delta N_2 = (K1*N_1*n_1)/(2*n_2) + (K1*N_2)*(n_1/n_2)$

Figure 3. Effort equations.

Table 2. Effort Equation Results

	Metric Value or Range			
Graph	N_2	n_1	N_1	n_2
δE vs. N_2	2350–2570	6	2565	269
E/N_2 vs. N_2	2350–2570	6	2565	269
E vs. n_1	2565	4–119	2565	269
$\delta E/\delta n_1$ vs. n_1	2565	4–119	2565	269
E vs. N_1	2565	5	1700–1800	2565
$\delta E/\delta N_1$ vs. N_1	2565	5	1700–1800	2565
E vs. n_2	2656	6	2565	14–269
$\delta E/\delta n_2$ vs. n_2	2656	6	2565	14–269

ing. Halstead (1975) states, "Nonetheless, ambiguous uses of operand names tend to reduce comprehensibility of a program . . ." (p. 41). The use of unique operand names tends to prevent or lessen multiple instantiations and the effects of coupling.

CONCLUSIONS AND LESSONS LEARNED

This article has aimed at showing the difficulty of associating metrics with KBSs. The example of the Effort metric has shown us that it is possible to develop metrics that can be used with KBSs. The results from the application of the software science metrics are not promising; the measures are not actively useful as the Effort values produced indicated. What does an Effort metric of 1,189,145.436 elementary mental discriminations actually mean? If we were to go on and develop an estimated value of the programming time by the equation:

$$T = E/S$$

where S is the Stroud number (Stroud, 1967) and which is usually set to 18 (Conte et al., 1986), we get a value equivalent to 764 days of work or 2.09 years: an estimate a little over 2 years longer than the system actually took to write!

Hence, it is clear that the system developer and project manager cannot use these metrics for assessing any measurement of effort in relation to KBSs. The answer may lie in more traditional directions such as a customized variation to COCOMO (Boehm, 1981), where systems engineers have had experience in developing historical databases of metrics for other complex software systems such as real-time embedded systems. Failure to understand the applicability of a series of metrics to an application development domain and their blind use can and will lead to mistakes and systems failure.

REFERENCES

Adamov, R., & Baumann, P. (1987). *Literature review on software metrics.* Institut fur Informatik, Der Universitat Zurich

Albrecht, A. J., & Gaffney, J. E. (1983). Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions of Software Engineering, SE-9*(6), 639–648.

Boehm, B. W. (1981). *Software engineering economics.* Englewood Cliffs, NJ: Prentice Hall.

Conte, S. D, Dunsmore, H. E, & Shen, V. Y. (1986). *Software engineering metrics and models.* Reading, MA: Benjamin Cummings.

Davidow, W. H., & Malone, M. S. (1992). *The virtual corporation.* New York: Harper Business Press.

DeSalvo, D. A., & Liebowitz, J. (1990). *Managing artificial intelligence & expert systems.* Yourdon Press Computing Series. Englewood, NJ: Prentice Hall.

Fenton, N. E. (1991). *Software metrics: A rigorous approach.* London: Chapman & Hall.

Forgy, C. L. (1981). *The OPS5 user's manual* (Tech Rep., CMU-CS-81-135). Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA.

Garmus, D., & Herron, D. (1996). *Measuring the software process.* Yourdon Press Computing Series. Upper Saddle River, NJ: Prentice Hall.

Grady, R. B., & Caswell, D. L. (1987). *Software metrics: Establishing a company wide program.* Englewood Cliffs, NJ: Prentice Hall.

Grady, R. (1992). *Practical software metrics and project management and process improvement.* Englewood Cliffs, NJ: Prentice Hall.

Halstead, M. H. (1975). *Elements of software science.* New York: Elsevier North Holland.

Hamer, P. G., & Frewin, G. D. (1982). M. H. Halstead's software science—a critical examination. *Proceedings of the 6th International Conference on Software Engineering* (pp. 197–206), Tokyo, Japan.

Harrison, W. A., Magel, K. I., Kluczny, R., & DeKock, A. (1982, September). Applying software complexity metrics to program maintenance. *IEEE Computer*, 65–79.

Henry, S., & Kafura, D. (1981). Software structure metrics based on information flow. *IEEE Transactions on Software Engineering, SE-7*(5), 510–518.

Kafura, D., & Reddy, G. R. (1987). The use of software complexity metrics in software maintenance. *IEEE Transactions on Software Engineering, SE-13*(3), 335–343.

King, M., & McAulay, L. (1995). A review of expert systems in management accountancy. *The New Review of Applied Expert Systems, 1.*

Liebowitz, J. (1991). *Institutionalizing expert systems.* Englewood Cliffs, NJ: Prentice Hall.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering, SE-2*(4), 308–320.

NASA. (1991). *CLIPS.* The Software Technology Branch, Information Technology Division, NASA/JSC.

O'Leary, D. E., Kukka, D., & Plant, R. T. (1997). Artificial intelligence and virtual organizations. *Communications of ACM, 40*(1), 52–59.

Pigford, D. V., & Baur, G. (1990). *Expert systems for business applications: Concepts and applications.* Boston: Boyd & Frazer.

Pressman, R. S. (1987). *Software engineering: A practitioner's approach.* New York: McGraw-Hill

Quinn, B. (1992). *Intelligent enterprise.* New York: Free Press.

SAIC. (1995). *Guidelines for the verification and validation of expert system software and conventional software* (EPRI TR-103331, Project 3093-01, Final Report). McClean, VA: Author.

Schneidewind, N. F., & Hoffmann, H. (1979). An experiment in software error data collection and analysis. *IEEE Transactions on Software Engineering, SE-5*(3), 276–286.

Shen, V. Y., Conte, S. D., & Dunsmore, H. E. (1983). Software science revisited: A critical analysis of the theory and its empirical support. *IEEE Transactions on Software Engineering, SE-9*(2), 155–165.

Sheppard, M. J. (1990). Design metrics: An empirical analysis. *Software Engineering Journal, 5*(1), 3–10.

Stroud, J. M. (1967). The fine structure of psychological time. *Annals of New York Academy of Science, 138*(2), 623–631.

Turban, E., & Liebowitz, J. (1992). *Managing expert systems.* Harrisburg, PA: Ideas Group Publishing.

Zuse, H. (1991). *Software complexity: Measures and methods* (pp. 143–169). New York: Walter de Gruyter.

Zolnowski, J. C., & Simmons, D. B. (1981). Taking the measure of program complexity. *Proceedings of the National Computer Conference*, 329–336.