Journal of
INDUSTRIAL
TECHNOLOGY

Volume 19, Number 1 - November 2002 to January 2003

A Method for Evaluating Expert System Shells for Classroom Instruction

By Dr. MD Salim, Mr. Alvaro Villavicencio, & Dr. Marc A. Timmerman

KEYWORD SEARCH

Administration
CAM
Computer Science
Information Technology
Management

Refereed Article

The Official Electronic Publication of the National Association of Industrial Technology • www.nait.org

A Method for Evaluating Expert System Shells for Classroom Instruction

By Dr. MD Salim, Mr. Alvaro Villavicencio, & Dr. Marc A. Timmerman

MD Salim holds a B.S. in Civil Engineering from the Bangladesh Institute of Technology, an M.S. in Construction Engineering from the University of Leeds (UK) and a Ph.D. in Civil Engineering from the North Carolina State University. He has served in numerous industrial positions and on the faculty of the University of Northern Iowa. His research interests and publications are in the applications of artificial intelligence to construction management.

Mr. Alvaro Villavicencio holds a B.S. in Industrial Engineering with a minor in Computer Sciences from the Catholic University of Chile, and a M.A.degree in Industrial Technology from the University of Northern Iowa. He is a Doctor of Industrial Technology Candidate at the University of Northern Iowa. He served as a faculty member at the University of Tarapacá, Arica-Chile, worked as consultant in Management Information Systems in different institutions in Chile, and also as a Staff Engineer in an Electric Distribution company in Chile. His research interests are in the areas of Artifical Intelligence Applications in Manufacturing and Construction, Expert Systems development, system integration, Robotics, and Automated Manufacturing.

Dr. Marc A. Timmerman holds a B.S.E.E. from the Santa Clara University, a M.Eng.E.E. from the Rensselaer Polytechnic Institute, and a Ph.D. from the George W. Woodruff School of Mechanical Engineering at the Georgia Institute of Technology. He has served on the faculties of the University of Tulsa, the Louisiana State University at Baton Rouge, the University of Arkansas at Little Rock, the University of Northern Iowa, and the Oregon Institute of Technology. His research publications are in the area of embedded microprocessors and DSP, Mechatronics, vibrations and magnetic bearings, optimization theory and artificial intelligence, and supply chain management.

Abstract

Contemporary Industrial Technology pedagogy includes the subject of Artificial Intelligence. Artificial Intelligence problems are often solved by students using commercial software packages called Expert Systems Shells. This paper describes the use of a method called Function Point Analysis for the evaluation of Expert System Shells for suitability in Industrial Technology pedagogy. Two types of Function Point Analysis are described in this paper; a direct method in which the Expert System Shell software itself is evaluated and an indirect method in which only the specification of the software is evaluated. The indirect method does not require the evaluator to have an actual copy of the software being evaluated, but it gives less useful results than the direct method, which does require that the evaluator have a copy of the software. Both methods are simple and straightforward and can be used successfully by Industrial Technology educators with no specialized background in Information Technology or Software Engineering.

Introduction

An Expert System is a computer program that can perform tasks using logic patterns similar to those used by a human expert. Expert systems can be constructed "from scratch" using standard programming languages like C/C++, Cobol, or other general-purpose computer languages. This is a laborious and lengthy process due to the complexity of the programs (Jackson, 1999). Expert System Shells are commercial software packages used to implement Artificial Intelligence projects in business, industry, government, and in academic programs training students for such activities. Using an Expert System Shell significantly reduces the time and labor of the creation of Artificial Intelligence programs by eliminating the need to program in a general-purpose programming language. A typical example of an Expert System program is a Decision Support System. A Decision Support System is a program that makes informed decisions based on human-like reasoning rules by evaluating various forms of data. A common application of Decision Support System is "fault identification." This is a program that helps unskilled workers to identify (and to fix!) the cause of equipment malfunctions by prompting the workers to make certain measurements and observations. The Decision Support System is programmed with the "rules" or knowledge of an expert worker in the maintenance of a particular type of equipment. The system helps an unskilled worker to fix a broken piece of equipment by using this program equipped with the expert worker's knowledge and experience without actually requiring the expert worker's presence. This type of Artificial Intelligence application allows a company to make the best possible use of its personnel resources and is of great economic interest in many sectors of business and industry.

Expert System Shell packages vary greatly in cost and performance, and in friendliness to the end user. They play an important role in the $250 million North American Artificial Intelligence software market. The National Academy Council (1999) has reported that over 12,000 Artificial Intelligence systems based on Expert System Shells are in use.

Purpose

An Industrial Technology Educator always wants to select a commercial Expert System Shell Package that is suitable for instruction. The instructor would like to first qualitatively group the available shells as to complexity. Next the instructor would like to numerically rank the acceptable shells into a list from most suitable to least suitable for instructional use. If there are ties in the ranking or if the best shells are unavailable due to cost, networking, hardware, or licensing issues, the instructor would like to be able to use combined qualitative and quantitative data to guide his or her decision.

The indirect method (qualitative assessment) described in this paper requires the instructor to obtain a description or specification of the software package to be studied. Based on these data, the instructor enumerates the software's features in five categories. These enumerations result in numerical values that are tabulated, multiplied by weights, and further arithmetically manipulated. The final results are a set of numerical values that describe the lines of code, number of man hours, etc., to write and debug a computer program in a high level programming language to implement the features of the Expert System Shell being studied. These numbers are an indirect measure of the relative complexity of the Shell being studied. For example, if Shell "A" yields a result that 50,000 lines of code would be needed to implement Shell "A's" function and Shell "B" yields a result that 20,000 lines of code would be needed to implement Shell "B's" function, then the instructor would infer that Shell "A" was more complex than Shell "B." The instructor can use these numbers to qualitatively rank the complexity of the shells being studied. In general the complexity of the shell gives an indication of the powerfulness, number of features, or problem solving strength of the shell.

The direct method (quantitative assessment) described in this paper requires the instructor to obtain a copy of the software to be evaluated and to perform a simple benchmark problem on this software. Based on his or her experience with the software, the instructor answers a set of nineteen questions about the software. The questions are quantitative and are based on a 0 (very false) to 5 (very true) numerical scale. After some arithmetic manipulation of these numbers, a single numerical factor results called the "Satisfaction Level" that ranges from 0 (least satisfied user) to 5 (most satisfied user) can be used to rank the software packages being studied as to their likelihood to be satisfactory to the end user, the student.

It should be noted that the direct and indirect methods could also be used independently. For example, if a small number of shells are to be tested and sample versions of all these shells are available, then the direct method can be used immediately. Likewise, if there are no sample versions available of the shells to be tested, the indirect method can be used independently.

Review Of Literature

As noted by Stylianou, Madey, and Smith (1992), the selection of an adequate Expert System Shell is often the difference between a successful and an unsuccessful industrial application. This observation is just as applicable to instructional uses of Expert Systems Shells as it is to industrial uses of such software packages.

Industrial Technology classes in Computer Integrated Manufacturing, Construction Management, Industrial Maintenance and Supervision, Occupational Health and Safety, and other areas often include computer laboratory exercises or projects based on implementing Decision Support Systems or some other type of Artificial Intelligence applications. These applications are often implemented on Expert System Shells. Some examples of such applications include the works of Koo and Aw (1992) and Roschke (1994) that have described applications of Decision Support Systems to construction management problems. El-Najdawi and Stylianou (1993) and Larsson (1996) have described the use of Decision Support Systems to problems in manufacturing and plant maintenance. Larsson (1996) describes a Decision Support System that uses an Expert System Shell to implement industrial maintenance on sensor systems in a manufacturing setting. Tait and Mehta (1997) give a detailed description of a Decision Support System called WORKBOOK that implements a complete chemical exposure analysis for industrial workplace chemical safety compliance. Xenakis (1990) gives a number of examples of the use of Expert System Shells to managerial/supervisory tasks.

According to two articles by Van Name and Catchings (1990), due to the large number of Expert System Shell packages on the market—and the lack of any standardized nomenclature for describing the features and capabilities of such products—selecting an appropriate package for an application can be very difficult even for experienced users. Head (1992) gives the sage advice to those contemplating the selection of an Expert System Shell to "think big" but to "start small." Current product reviews of Expert System Shells in the software trade literature are of very limited use as they assume that the reader is already familiar with the current version of the shell and the author or authors only provide details of changes and improvements to that shell's latest version. In order to get details of the basic structure of the shell it is often necessary to go to archival sources like Pallatto (1991), Rasmus (1989), Coleman (1989), Shepherd (1988), who respectively describe the basic functions of first-generation Expert System Shell packages from IBM, Nexpert, Xi-Plus, and Texas Instruments.

Unfortunately, other than an excellent article by Moffitt (1994), there is not much literature on the selection of Expert System Shells from a purely pedagogical perspective. Nevertheless, Stylianou, Madey, and Smith (1992) identify a total of eight (8) categories of criteria for evaluating Expert System Shells; End User Interface, Developer Interface, System Interface, Inference Engine, Knowl-

edge Base, Data Interface, Cost, and Vendor-Related aspects.

From an industrial and business user perspective, Plant and Salinas (1994) and Dudman (1993) have presented a "bench-mark" based framework for the evaluation of Expert System Shell packages. Their general concept is that a standard" bench-mark" problem is formulated and run on various shells and the results are compared for time to execute, correctness of results, ease of use, and so forth. Unfortunately, the outcome of this assessment is very dependent on the specific "bench-mark" problem chosen and this method is thus very limited in its usefulness. Murrell and Plant (1997) and Antoniou, van Harmelen, and Plant (1998) have published surveys of more sophisticated methods for software assessment that attempt to avoid this limitation of the "bench-mark" method. Such methods include the "constructive cost model" of Boem (1981) and the method of Putnam and Meyers (1992). Putnam and Meyers champion the concept of "excellence" measures that deal quantitatively with issues like maintainability and user friendliness of software packages. Both of these methods are based on using the size of the program, the labor involved in writing the program, and other such measures of complexity and effort, to assess the software's general usability and overall quality. These methods, though simple to implement, do not give a good idea of the suitability of a software package for a *particular* use, but do give a *general view* of the software's overall complexity and friendliness. Wentworth, Knaus, and Aougab (1999), and Pressman (1997) have presented approaches based on statistical concepts that give good estimates of the likelihood that a software package will be powerful and accurate, i.e., *give good answers*. Unfortunately, these methods do not give much insight as to the *difficulty or friendliness* of the steps leading to that answer. Finally, Banyahia (1996) gives an excellent summary of these issues in a format useful to Industrial Technology education professionals. As can be seen from these references, the key problem is finding a single method that assess *both the power/accuracy and the user friendliness* of an Expert System Shell.

Function Point Analysis is a methodology for assessing a software package that can measure both the accuracy and the user friendliness of an Expert System Shell. The Function Point Analysis method is based on evaluating specific features of the software package on a numerical scale, multiplying these numbers by weights that assigns the relative importance of each feature, and formulating a numerical figure of merit for the package as a whole by adding these weight and evaluation products. A final scaling factor is often used to bring the answer into an easy-to-interpret range of 0 to 5 points or 0 to 10 points—with 0 representing poor satisfaction. According to Cheung, Willis, and Milne, (1999) the Function Point Analysis method originated with Albrecht in 1979 and has become very popular among Information Technology specialists. There is a large literature on methods to apply Function Point Analysis to particular types of problems in software assessment. Such literature includes work by Robillard (1996), and Jeffrey and Low (1993). A paper by Furey (1997) aptly summarizes the benefits of the Function Point Analysis approach:

> *Although they are not a perfect metric, they have proven to be very useful. Among the desirable features of function points are technology independence, consistency and repeatability, data normalization and estimating and comparing. (p. 28)*

The methodology presented in this study is based on Function Point Analysis. The work of Boem (1981) and Putnam and Meyers (1992) is used to augment the Function Point Analysis method with other advanced concepts.

Function point analysis has become a standard, mature technique widely used by industrial practitioners. Biderman (1990) mentions that an international society dedicated to function point analysis, the International Function Point User's Group (IFPUG), was founded in 1986 and had enrolled over 200 members. As function point analysis is a mature technique, there are relatively few journal articles about function point analysis in the current experimental literature. A recent article by Cheung, Willis, and Milne (1999) describes many ongoing industrial applications of function point analysis. Another recent article by Orr and Reeves (2000) describes a pedagogical framework for teaching function point analysis in a Business program.

Methodology

Two types of evaluation data can be generated for an Expert System Shell package, a *direct method* and *an indirect method*.[1] The first type of data is "Satisfaction Level" which is a *direct measure* of the overall user satisfaction with the shell. For example, if a sample copy or demo copy of the Expert System Shell software is available, the evaluator will run an example problem. Based on his / her experiences with the software, the evaluator can calculate a numerical "Satisfaction Level" based on his / her experience with the shell. Many vendors of Expert Systems Shells do provide free demo or sample limited-use versions of their products to allow potential buyers to perform this type of study.

Direct Method Methodology

Table 1 presents the direct method test instrument. This instrument is completed by the evaluator as follows:

(1) The evaluator obtains demonstration or sample copies of the software packages to be evaluated.

(2) The evaluator selects a bench-mark problem, for example a text book example problem, and runs this problem on each of the software packages.

[1] *The methods presented in this study are specific applications of very general methods described in a seminal text by Pressman (1997). Readers interested in the scientific/statistical bases of these methods are referred to this most excellent general treatment of this subject matter.*

(3) After running the bench-mark problem, the evaluator responds to the 19 questions in the instrument and estimates a quantitative answer to each question on a 0 to 5 scale with 5 being very true and 0 being very false.
(4) Each numerical result is multiplied by a weighting factor as given in the weight column.
(5) The weighted values are summed and then divided by 26 (as for 19 questions, 7 questions have weight factor of 2 and 12 questions have weight factor of 1, i.e., 7x2 + 12x1= 26) to give a result in the numerical range of 0 to 5.

Three complete numerical examples of the direct method and a paradigm for interpreting the direct method result are presented in the Findings section.

Indirect Method Methodology

The second method is based on estimating the resources needed to write a program using general computer languages to implement an Expert System. This type of calculation is based on finding answers to the following questions, "If a user wrote a program in some computer language to solve this problem instead of using a ready-made Expert System shell, how long would this program be, how many person-months would it take to write this program, how many errors might this program have, and how many test cases might the user need to run to debug the program?" This question assumes that LISP or PROLOG, two common languages used for Artificial Intelligence programs, are being used for this hypothetical equivalent computer program.

Table 2 presents the indirect method test instrument.[2] This instrument is used as follows:

(1) The evaluator obtains specifications or other descriptive materials for the software packages to be evaluated.
(2) The evaluator enumerates the number of Internal Knowledge Structures, External Data, Input Decisions, Decisions Supported at Output, and User Inquiries supported in the software packages.
(3) The evaluator then estimates the complexity level of each of these features as Low, Medium, or High, and selects a numerical weight based on this complexity.
(4) The evaluator multiples the enumerated values in step 2 by the weights in step 3 and sums the weighted value. This number

Table 1. Direct measurement test instrument.

Category	Question	Assessment	Weight	Value x Weight
Correctness of Answer	Is there enough information to evaluate the software?		**2**	
	Does the software give the same answers as other methods of calculation?		**2**	
	Does the software give the same answer that a human expert would give?		**2**	
	Does the software provide the right answer for the right reasons?		**2**	
Accuracy of Answer	Is the software accurate in its answer(s)?		**2**	
	Is the answer complete? Does the user need to do additional work to get a usable result?		**2**	
	Is the procedure of getting the answer simple and dear?		**2**	
Correctness of Reasoning Technique	Does the answer change if new but irrelevant data is entered into the software?		**1**	
	Can the system clearly explain its reasoning technique to the user?		**1**	
	Does the system require a lot of irrelevant questions to reach the answer?		**1**	
Sensitivity	Does the answer change if irrelevant changes are made to the system's rules?		**1**	
Reliability	Does the software crashes or hang-ups in its host computer?		**1**	
	Does the system give warnings for cases involving incomplete data or rules?		**1**	
Cost Effectiveness	Is the cost of the system justified by its performance?		**1**	
	Does the shell have all the features listed in the vendor's literature?		**1**	
	Does the software still provide answers with incomplete knowledge?		**1**	
Limitations	Can limitations of the shell be detected at this point in time?		**1**	
	Does the shell allow the user to expand a program if needed?		**1**	
	Can the system learn from increased data or experience?		**1**	
Results	Add Weight x Value Divide by 26			
		5-4-3-2-1-0 True…False		5 = Satisfied User 0=Unsatisfied User

[2] *The methodology of this instrument is adapted from Pressman (1997). Pressman's results are based on statistical analyses of numerous software projects and "curve fitting" techniques to derive simple algebraic approximations based on these studies. The weighing factors in this instrument come from these statistical analyses.*

is referred to as the "Function Point" (singular).

(5) Finally, by division "/", multiplication "x", and exponentiation"^", of this "Function Point" (singular) numerical value, the evaluator calculates other "Function Points" (plural) that are the end result of the indirect method:
   (a) Equivalent Lines of Code Needed to Solve Same Problem = (Function Points) x 64
   (b) Number of Test Cases Needed to Debug Equivalent Code = (Function Points) ^ 1.2
   (c) Number of Probable Defects/ Bugs in Equivalent Code = (Function Points) ^ 1.25
   (d) Number of Months to Write Equivalent Code = (Function Points) ^ 0.4
   (e) Number of Persons Needed to Write Equivalent Code = (Function Points)/150

(6) One value requires a further calculation, the number of Person-Months needed is the product of the numerical values of the number of persons and the number of months: Total Person-Months for Equivalent Code Project = Number of Months x Number of Persons

These results of the indirect method do not have a simple interpretation. These numbers estimate the scope of an equivalent LISP / PROLOG language programming project with the same functionality as the Expert System Shell being evaluated. There is no direct interpretation of these numbers as an end user's potential satisfaction, but these numbers can be used to rank and compare the potential complexity of an Expert System Shell product in the situation where a version of the product is not available for evaluation. Typically, the assumption is made that a more complex package has more features or functions and is therefore is more powerful. The Findings section illustrates a possible approach to interpreting these data.

Some Practical Issues with this Methodology

If a very large number of software packages are to be evaluated, the less labor intensive indirect method can be used for a preliminary ranking of the software packages and the more time intensive direct method can be used to refine the selection process once a subset of packages has been identified for a further and more detailed study.

These methods are based on standard techniques with which all Information Technology managers are familiar. In situations where software purchases require administrative approval, studies of this nature can be very helpful in obtaining administrative approval from Information Technology managers for making this type of software purchase.

A further use of the indirect method is in estimating the amount of lecture and lab time needed to cover the basic functions of the Expert Systems Shell. In the situation where the Expert System Shell will be one of several elements covered in a class, there may be a strong advantage in favoring a simpler product over a more complex product.

Findings

Test Cases Presented

To illustrate the use of this paradigm, a complete instructional usability analysis of three common Expert System Shells products is presented:

(1) MP2 Professional version 5.0 from Datastream Corporation of Greenville, South Carolina, USA. Table 3 is a direct analysis, Table 4 is an indirect analysis.

(2) EXSYS CORVID from Albuquerque, New Mexico, USA. Table 5 is a direct analysis, Table 6 is an indirect analysis.

(3) Art * Enterprise from Mindbox of Greenbrae, California, USA. Table 7 is a direct analysis, Table 8 is an indirect analysis.

Table 2. Indirect measurement test instrument.

Program Capability	Program Count	Complexity Level Weight Low	Medium	High	Weight Selected	Count x Weight
Internal Knowledge Structures		**7**	**10**	**15**		
External Data		**5**	**7**	**10**		
Input Decisions		**3**	**4**	**6**		
Decisions Supported at Output		**4**	**5**	**7**		
User Inquiries		**3**	**4**	**6**		
Function Points for Indirect Method = Sum of Weight Selected x Program Count						

Indirect Program Measures		
Equivalent Lines of Code Needed to Solve Same Problem	(Function Points) x 64	
Number of Test Cases Needed to Debug Equivalent Code	$(\text{Function Points})^{1.2}$	
Number of Probable Defects/Bugs in Equivalent Code	$(\text{Function Points})^{1.25}$	
Number of Months to Write Equivalent Code	$(\text{Function Points})^{0.4}$	
Number of Persons Needed to Write Equivalent Code	(Function Points)/150	
Total Person-Months for Equivalent Code Project	Number of Months x Number of Persons	

In assessing the Expert System Shells, it is important to keep in mind that the complexity of using the shell is strongly influenced by the complexity of the problem the shell is being used to solve. It would not be meaningful to compare "Shell A" solving a simple problem to "Shell B" solving a complicated problem. The shells should be evaluated consistently on a single benchmark problem to insure that their features needed to solve non-trivial, complex problems are being uniformly evaluated. In this study, a simple problem of the scheduling part flow in a machine shop was used as the benchmark problem.

The direct method presents an unexpected problem in that the demo versions of Expert System Shell packages commonly made available by vendors are often deeply scaled down versions that are missing many of the features of the full commercial version of the software package. Care must be taken to carefully research the features of the full commercial software package and perform a consistent and meaningful analysis.

Interpretation of the Data of the Test Cases

The italic numerals in the Tables 3 through 8 represent values entered by the person evaluating the shells. Table 9 is a summary of the results of Tables 3 through 8. The values are rounded for ease of comparison. The direct method calculations shown in Tables 3, 5, and 7, give an estimation of end-user Satisfaction Level. The indirect calculations in tables 4, 6, and 8, give estimates of the relative complexity of the packages. Table 9 presents the direct and indirect results for comparison purposes.

The evaluator must use some thought and insight to interpret the results. The results seem to group the shells into two categories. The MP2 shell has a high satisfaction number at 4.54 but also has a high code line count at 15,616. The EXSYS CORVID and the Art * enterprise shells seem to lump together as they both had similar satisfaction numbers at 4.35 and 4.31 and much lower code line numbers at 6,272 and 9,472. The data would suggest that there is a trade-off between the complexity of a program, the ease of use of a program, and the power of a program. Figure 1 depicts a graphical interpretation of these data. Based on the direct results, all three packages show the potential to satisfy the user. However, the MP2 package is a much more complex software package than the Art * Enterprise package or the EXSYS CORVID package. This would suggest that MP2 is both easy to use and has many features while Art * Enterprise and EXSYS CORVID are also easy to use but have fewer features. Based on this interpretation of the data, the instructor might have the following ranking:

(1) Purchase MP2 as the most desirable package.
(2) Purchase either Art * Enterprise or EXSYS CORVID as a second choice. As both of these packages have similar direct and indirect rankings, the choice of which package to purchase should be based on cost.

Table 3. MP2 Results. Direct measurement technique.

Category	Question	Assessment	Weight	Value x Weight
Correctness of Answer	Is there enough information to evaluate the software?	*5*	**2**	*10*
	Does the software give the same answers as other methods of calculation?	*5*	**2**	*10*
	Does the software give the same answer that a human expert would give?	*5*	**2**	*10*
	Does the software provide the right answer for the right reasons?	*5*	**2**	*10*
Accuracy of Answer	Is the software accurate in its answer(s)?	*5*	**2**	*10*
	Is the answer complete? Does the user need to do additional work to get a usable result?	*5*	**2**	*10*
	Is the procedure of getting the answer simple and dear?	*5*	**2**	*10*
Correctness of Reasoning Technique	Does the answer change if new but irrelevant data is entered into the software?	*5*	**1**	*5*
	Can the system clearly explain its reasoning technique to the user?	*3*	**1**	*3*
	Does the system require a lot of irrelevant questions to reach the answer?	*5*	**1**	*5*
Sensitivity	Does the answer change if irrelevant changes are made to the system's rules?	*5*	**1**	*5*
Reliability	Does the software crashes or hang-ups in its host computer?	*5*	**1**	*5*
	Does the system give warnings for cases involving incomplete data or rules?	*5*	**1**	*5*
Cost Effectiveness	Is the cost of the system justified by its performance?	*5*	**1**	*5*
	Does the shell have all the features listed in the vendor's literature?	*5*	**1**	*5*
	Does the software still provide answers with incomplete knowledge?	*1*	**1**	*1*
Limitations	Can limitations of the shell be detected at this point in time?	*5*	**1**	*5*
	Does the shell allow the user to expand a program if needed?	*4*	**1**	*4*
	Can the system learn from increased data or experience?	*0*	**1**	*0*
Results	Add Weight x Value Divide by 26			118 4.54
		5-4-3-2-1-0 True…False		5 = Satisfied User 0=Unsatisfied User

Implications In The Research

Expert System Shell software assessment studies have appeared in many computer-related journals and trade publications. Articles in these publications tend to focus on one or more specific packages available at that time and gives one an overall impression of the shell, but provide no definite information on performance (Stylianou, Madey, and Smith, 1992). Assessment methods that give a useful measure of the user complexity and computational power of such software packages are also described in great detail in the literature. There is unfortunately no single method that gives a measure of both the power and the complexity of a software package as a unified measurement. A further limitation is that these methods in the literature do not do a good job of narrowing this evaluation to a *specific*

Table 4. MP2 Results. Indirect measurement technique.

Program Capability	Program Count	Complexity Level Weight Low	Medium	High	Weight Selected	Count x Weight
Internal Knowledge Structures	*6*	**7**	**10**	**15**	*15*	90
External Data	*5*	**5**	**7**	**10**	*10*	50
Input Decisions	*5*	**3**	**4**	**6**	*3*	15
Decisions Supported at Output	*8*	**4**	**5**	**7**	*7*	56
User Inquiries	*11*	**3**	**4**	**6**	*3*	33
Function Points for Indirect Method = Sum of Weight Selected x Program Count						244

Indirect Program Measures		
Equivalent Lines of Code Needed to Solve Same Problem	(Function Points) x 64	15616
Number of Test Cases Needed to Debug Equivalent Code	(Function Points) $^{1.2}$	733
Number of Probable Defects/Bugs in Equivalent Code	(Function Points)$^{1.25}$	964
Number of Months to Write Equivalent Code	(Function Points)$^{0.4}$	9
Number of Persons Needed to Write Equivalent Code	(Function Points)/150	1.63
Total Person-Months for Equivalent Code Project	Number of Months x Number of Persons	15

Table 5. EXSYS CORVID example. Direct measurement technique.

Category	Question	Assessment	Weight	Value x Weight
Correctness of Answer	Is there enough information to evaluate the software?	*4*	**2**	*8*
	Does the software give the same answers as other methods of calculation?	*5*	**2**	*10*
	Does the software give the same answer that a human expert would give?	*5*	**2**	*10*
	Does the software provide the right answer for the right reasons?	*5*	**2**	*10*
Accuracy of Answer	Is the software accurate in its answer(s)?	*5*	**2**	*10*
	Is the answer complete? Does the user need to do additional work to get a usable result?	*5*	**2**	*10*
	Is the procedure of getting the answer simple and dear?	*4*	**2**	*8*
Correctness of Reasoning Technique	Does the answer change if new but irrelevant data is entered into the software?	*5*	**1**	*5*
	Can the system clearly explain its reasoning technique to the user?	*2*	**1**	*2*
	Does the system require a lot of irrelevant questions to reach the answer?	*2*	**1**	*2*
Sensitivity	Does the answer change if irrelevant changes are made to the system's rules?	*4*	**1**	*4*
Reliability	Does the software crashes or hang-ups in its host computer?	*5*	**1**	*5*
	Does the system give warnings for cases involving incomplete data or rules?	*3*	**1**	*3*
Cost Effectiveness	Is the cost of the system justified by its performance?	*3*	**1**	*3*
	Does the shell have all the features listed in the vendor's literature?	*4*	**1**	*4*
	Does the software still provide answers with incomplete knowledge?	*5*	**1**	*5*
Limitations	Can limitations of the shell be detected at this point in time?	*4*	**1**	*4*
	Does the shell allow the user to expand a program if needed?	*5*	**1**	*5*
	Can the system learn from increased data or experience?	*5*	**1**	*5*
Results	Add Weight x Value			113
	Divide by 26			4.35
		5-4-3-2-1-0 True…False		5 = Satisfied User 0=Unsatisfied User

Table 6. EXSYS CORVID example. Indirect measurement technique.

Program Capability	Program Count	Complexity Level Weight Low	Medium	High	Weight Selected	Count x Weight
Internal Knowledge Structures	*2*	**7**	**10**	**15**	*15*	30
External Data	*0*	**5**	**7**	**10**	*0*	0
Input Decisions	*4*	**3**	**4**	**6**	*6*	24
Decisions Supported at Output	*2*	**4**	**5**	**7**	*7*	14
User Inquiries	*10*	**3**	**4**	**6**	*3*	30
Function Points for Indirect Method = Sum of Weight Selected x Program Count						98

Indirect Program Measures		
Equivalent Lines of Code Needed to Solve Same Problem	(Function Points) x 64	6272
Number of Test Cases Needed to Debug Equivalent Code	$(\text{Function Points})^{1.2}$	245
Number of Probable Defects/Bugs in Equivalent Code	$(\text{Function Points})^{1.25}$	308
Number of Months to Write Equivalent Code	$(\text{Function Points})^{0.4}$	6
Number of Persons Needed to Write Equivalent Code	(Function Points)/150	0.65
Total Person-Months for Equivalent Code Project	Number of Months x Number of Persons	4

Table 7. Art * Enterprise example. Direct measurement technique.

Category	Question	Assessment	Weight	Value x Weight
Correctness of Answer	Is there enough information to evaluate the software?	*5*	**2**	*10*
	Does the software give the same answers as other methods of calculation?	*5*	**2**	*10*
	Does the software give the same answer that a human expert would give?	*5*	**2**	*10*
	Does the software provide the right answer for the right reasons?	*5*	**2**	*10*
Accuracy of Answer	Is the software accurate in its answer(s)?	*4*	**2**	*8*
	Is the answer complete? Does the user need to do additional work to get a usable result?	*4*	**2**	*8*
	Is the procedure of getting the answer simple and dear?	*4*	**2**	*8*
Correctness of Reasoning Technique	Does the answer change if new but irrelevant data is entered into the software?	*4*	**1**	*4*
	Can the system clearly explain its reasoning technique to the user?	*3*	**1**	*3*
	Does the system require a lot of irrelevant questions to reach the answer?	*5*	**1**	*5*
Sensitivity	Does the answer change if irrelevant changes are made to the system's rules?	*4*	**1**	*4*
Reliability	Does the software crashes or hang-ups in its host computer?	*4*	**1**	*4*
	Does the system give warnings for cases involving incomplete data or rules?	*3*	**1**	*3*
Cost Effectiveness	Is the cost of the system justified by its performance?	*5*	**1**	*5*
	Does the shell have all the features listed in the vendor's literature?	*5*	**1**	*5*
	Does the software still provide answers with incomplete knowledge?	*4*	**1**	*4*
Limitations	Can limitations of the shell be detected at this point in time?	*4*	**1**	*4*
	Does the shell allow the user to expand a program if needed?	*4*	**1**	*4*
	Can the system learn from increased data or experience?	*3*	**1**	*3*
Results	Add Weight x Value Divide by 26			112 4.31
		5-4-3-2-1-0 True…False		5 = Satisfied User 0=Unsatisfied User

application (like instructional use) of these software packages.

The method presented in this paper is a specialization of the popular Function Point Method for software assessment applied to the particular problem of selecting an Expert System Shell for Industrial Technology pedagogical use. The direct form of this method gives a simple numerical indication of the likely satisfaction level of the software's end users, the students. Although this method is based on a subjective evaluation technique, the results do provide a useful scale for selecting Expert System Shells based on a common characteristic. The indirect form of this method is useful for estimating the power / complexity of Expert System Shell packages. This indirect technique does not require access to the actual software package and is less labor intensive than the direct technique. Together, the direct and indirect techniques allow for an estimate of both the power / complexity and the user satisfaction / friendliness

Table 8. Art * Enterprise example. Indirect measurement technique.

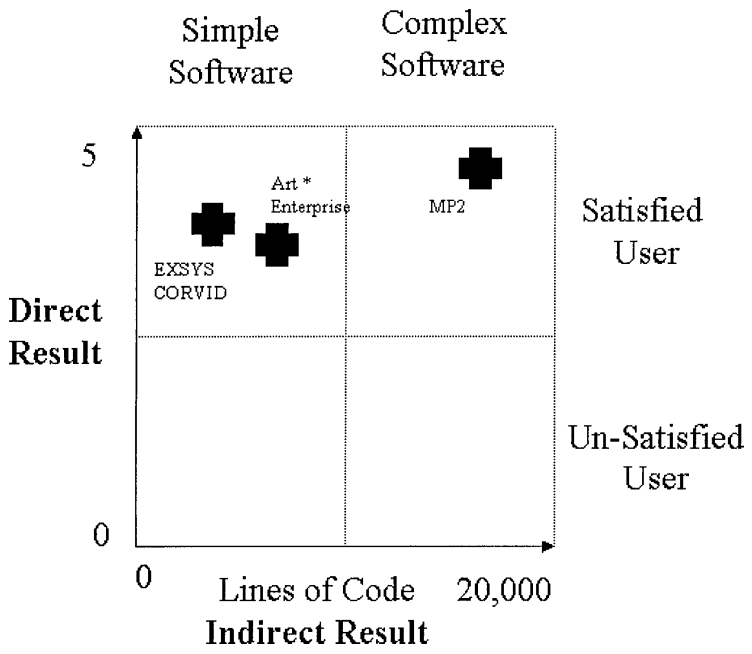
Program Capability	Program Count	Complexity Level Weight Low	Medium	High	Weight Selected	Count x Weight
Internal Knowledge Structures	*6*	**7**	**10**	**15**	*15*	90
External Data	*3*	**5**	**7**	**10**	*7*	21
Input Decisions	*4*	**3**	**4**	**6**	*4*	16
Decisions Supported at Output	*3*	**4**	**5**	**7**	*7*	21
User Inquiries	*0*	**3**	**4**	**6**	*0*	0
Function Points for Indirect Method = Sum of Weight Selected x Program Count						148

Indirect Program Measures		
Equivalent Lines of Code Needed to Solve Same Problem	(Function Points) x 64	9472
Number of Test Cases Needed to Debug Equivalent Code	$(\text{Function Points})^{1.2}$	402
Number of Probable Defects/Bugs in Equivalent Code	$(\text{Function Points})^{1.25}$	516
Number of Months to Write Equivalent Code	$(\text{Function Points})^{0.4}$	7
Number of Persons Needed to Write Equivalent Code	(Function Points)/150	0.99
Total Person-Months for Equivalent Code Project	Number of Months x Number of Persons	7

Table 9. Results for all three cases presented.

Expert System Shell	Direct Method Results	Indirect Method Results				
		Function Points	Equivalent lines of code	Test cases for debug	Number of defects/bugs	Effort (Man-Month)
MP2	4.54	244	15,616	733	964	15
EXSYS CORVID	4.35	98	6,272	245	308	4
Art * Enterprise	4.31	148	9,472	402	516	7

Figure 1. Interpretation of Direct and Indirect Results.



of Expert System Shells. *For an Expert System Shell package to be pedagogically useful, it must be both powerful and friendly.*

Debate, discussion, and literature on this important pedagogical problem is lacking and it is hoped that this paradigm, as imperfect as it is, will spark an interest for study and scholarly discourse on this very practical problem in the Industrial Technology Community. Artificial Intelligence and Decision Support Systems are rapidly entering the mainstream of industrial practice and as such will continue to be of interest to those responsible for planning and implementing the Industrial Technology Curriculum.

References

Albrecht, A. (1979). Measuring applications development productivity. Proceedings of the Joint Share/ Guide/IBM Application Development Symposium, 83-92.

Antoniou, G., van Harmelen, F., & Plant, R. T. (1998, Fall). Verification and validation of knowledge-based systems: Report on two 1997 events. AI Magazine, 19(3), 123-126.

Biderman, B. (1990). Using function Points. Computing Canada, 16(4), 30-32.

Boem, B. (1981). Software Engineering Economics. New York, NY: Prentice Hall.

Banyahia, H. (1996, Spring). Costs and productivity estimation in computer engineering economics. Engineering Economist, 41(3), 229-242.

Cheung, Y., Willis, R. & Milne, B. (1999). Software benchmarks using function point analysis. Benchmarking: An International Journal, 6(3), 269-279.

Coleman, T. (1989, March 29). Expert-ease. PC User, 94-96.

Furey, S. (1997, March-April). Why we should use function points. IEEE Software, 14(2), 28-30.

Dudman, J. (1993, May 13). Measure of success. Computer Weekly, 40-41.

El-Najdawi, M. K., & Stylianou, A. C. (1993, December). Expert Support Systems: Integrating AI technologies. Communications of the ACM, 36, 54-65.

Head, R. V. (1992, April 13). Planning an expert system? Think big, but start small. Government Computer News, 11(8), 23-24.

Koo, T. K., & Aw, Y. B. (1992, May). Using expert systems to manage professional survey practices. Journal of Surveying Engineering 118(2), 43-63.

Jackson, P. (1999). Introduction to Expert Systems. Harlow, England: Addison-Wesley.

Jeffery, D. R., & Low, G. C. (1993, May). A comparison of function point counting techniques. IEEE Transactions on Software Engineering, 19(5), 529-533.

Larsson, J. E. (1996, January). Diagnosis based on explicit means-end models. Artificial Intelligence, 80, 29-93.

Moffitt, K. (1994, May-June). An analysis of the pedagogical effects of expert system use in the classroom. Decision Sciences, 25(3), 445-461.

Murrell, S., & Plant, R. T. (1997, December). A survey of tools for the validation and verification of knowledge-based systems: 1985-1995. Decision Support Systems, 21, 302-323.

National Academy Council. (1999). Funding a revolution. Washington D.C.: National Academy Press.

Orr, G., & Reeves, T. E. (2000). Function point counting: one programs experience. The Journal of Systems and Software, 53, 239-244.

Pallatto, J. (1991, July 29). Expert tools take the spotlight; IBM, AICorp stress graphical interfaces, ease of use. PC Week 8(30), 55-56.

Plant, R. T., & Salinas, J. P. (1994, August). Expert systems shell benchmarks: The missing comparison factor. Information and Management, 27, 89-101.

Pressman, R. (1997). Software Engineering: A Practitioner's Approach. New York, NY: McGraw-Hill.

Putnam, L., & Myers, W. (1992). Measures for Excellence. Englewood Cliffs, NJ: Prentice Hall / Yourdon Press.

Rasmus, D. (1989, September). The expert is in. MacUser 5(9), 136-147.

Robillard, P. N. (1996, December). Function points analysis: An empirical study of its measurements processes. IEEE Transactions on Software Engineering, 22(12), 895-901.

Roschke, P. N. (1994, September-October). Validation of knowledge-based system with multiple bridge rail experts. Journal of Transportation Engineering, 120, 787-806.

Schmuller, J. (1999). Sams teach yourself UML in 24 hours. Indianapolis, IN: Sams Publishing.

Shepherd, S. J. (1988, July). Sophisticated expert. PC Tech Journal July 6(7), 106-117.

Stylianou, A. C., Madey, G. R., & Smith, R. D. (1992). Selection criteria for Expert System shells: A socio-technical framework. Communications of the ACM, 35 (10), 30-48.

Tait, K., & Mehta, M. (1997, August). Validation of the workplace exposure assessment expert system WORKBOOK. American Industrial Hygiene Association Journal, 58, 592-602.

Van Name, M. L., & Catchings, B. (1990, March 7). Choices abound in expert-system shells. PC Week, 7(9), 77-79.

Van N, M. L., & Catchings, B. (1990, March 7). Expert-system shell users advice forethought. PC Week, 7(9), 78-79.ame

Wentworth, J., Knaus, R., & Aougab, H. (1999). FHWA Handbook: Verification, Validation, and Evaluation of Expert Systems. McLean, VA: Turner-Fairbank Highway Research Center.

Xenakis, J. J. (1990, September 3). Real use of AI: expert systems find widespread use in mainstream applications - finally. InformationWeek, 30-32.