J. King Saud Univ., Vol. 6, *Computer & Information Sci.*, pp. 71-83 (A.H 1414/1994)

A Formal Method of Evaluating Effectiveness of Computer Science Curriculum

Aftab Ahmad

Department of Computer Science, College of Computer and Information Sciences, King Saud University, P.O. Box 51178, Riyadh 11543, Saudi Arabia

(Received 15 April 1992; accepted for publication 27 October 1992)

Abstract. Computer Science B.S. Curricula have been designed by various expert committees during the current decade. Many experts have subsequently analyzed the effectiveness of the introductory Computer Science courses. These analysis have indicated that the first course in Computer Science is quite successful. There is a need for techniques and tools to measure objectively the quality of programming level achieved by undergraduate Computer Science students in the course of their degree completion. We have applied the well-established theories of software science metrics on volumes of data-collected to verify the effectiveness of the complete Computer Science B.S. Curricula. It has been found that the curricula have by far achieved their objective. The proposed method and data-collected will be a step forward towards giving insight into objective evaluation of Computer Science curricula. This should also be useful to curriculum designers and university managers who are looking for "Outcomes Assessment" of their Computer Science degree programs. Furthermore, the work reported here provides a good means for verification of software science metrics.

1. Introduction

Automation of manufacturing is proceeding at a fast pace all over the world. Automation involves the computerization of the design, the production, and the testing of required products. Thus, the future industrial engineers and managers must be computer-literate.

Training of fresh graduates and retraining of industrial employees are performed by colleges with Computer Science expertise. Invariably the courses of B.S. Computer Science are opened to industrial employees during the evening hours. Accordingly, the Computer Science courses have a major impact on future of industrial activities.

Recognizing the importance of Computer Science education, many expert committees have designed appropriate curricula [1-18]. These curricula are followed in

many schools of Computer Science. Of course, some colleges do modify the above mentioned curricula to satisfy local conditions and other special requirements. Many experts involved in Computer Science education have recognized the importance of implementing the above curricula recommendations effectively. A number of experiments on the effectiveness of individual Computer Science courses have been conducted. It is well known that the first course in any discipline sets the tone for the more advanced courses to follow. Thus, most of the research of effectiveness of teaching has been concerned with introductory Computer Science courses [19-28].

ACM as well as IEEE reports [1-6] have recommended that the undergraduate Computer Science curricula should be mainly concerned for the development of "quality programming" rather than just programming skills. There is a need for tools to measure objectively the quality programming levels achieved by undergraduate students in the course of their degree completion.

It is our belief that most of the colleges following the expert designed Computer Science curricula would be graduating their initial batches of students. It is the natural and logical next step to concern ourselves with the effectiveness of the complete curricula at this stage. The paper proposes to study software science metrics to estimate effectiveness of a given academic program towards developing quality programming skills, as the student moves from the freshman year to the senior years.

We have been observing and collecting data on aquisition of programming skills for the last 6 years from the computer programs of undergraduate Computer Science majors (ranging from Freshmen to Seniors) at the College of Computer and Information Sciences of King Saud University. The five-year B.S. Computer Science curriculum was designed here during 1983-84 based on recommendations [2-6] and is substantially more rigorous and demanding than ACM curriculum [2]. Fortunately, our Computer Science B.S. program also satisfies largely recommendations of current ACM curriculum [1]. In the curriculum followed by us the following important courses are considered for evaluation and data collection.

The courses are:

1. Computer Programming-I
2. Computer Programming-II
3. Data Structures
4. File Processing Techniques
5. Operating Systems-I
6. Compiler Design
7. Software Engineering
8. Operating System-II
9. Computer Networks
10. Graduation Projects.

It can be observed that the above ten courses are the backbone of most of the curricula followed [1-6].

While the growing cost and complexity of software has increased the need for objective software metrics, there is an equal need for collection and analysis of

empirical data on a variety of implementations of the same algorithms to support the theoretical research in software science metrics. There is an excellent opportunity of such data collection and analysis in the academic environments at universities that offer degree in Computer Science. The Computer Science departments can be viewed as on-going computer-program production facilities where student-programmers produce computer programs in a large variety of problem areas during the course of their study. Because different participant students go through the same courses, programming essentially the same classes of problems, a large number of implementations of the same algorithms become available -- a facility not economical in other software industry environments. Furthermore, the data collected in this type of environment is most ideal for verification of metrics like intelligent content and language levels. We expect that work presented here provides an excellent opportunity for verification of software science metrics.

In section 2, we briefly describe the software science theory which forms the basis for evaluating the effectiveness of a Computer Science curriculum. The next section details the data-collection and development tools so that the measures of software science theory can be calculated. Section 4 presents the results of analysis. The last section analyzes the results and concludes the paper.

2. The Concept

Halsteads' [29] theory of software science metrics is intellectually appealing. These metrics have been widely accepted and applied extensively. A detailed discussion of software science metrics can be found in [29,30]. In this section we briefly introduce some of the relevant metrics.

Halsteads' [29] has defined a number of impurities which reduce the effectiveness of a program by increasing the length of the program unnecessarily. The parameter $\hat{N}$ (estimated length) defined by Halstead is an estimate of a length of a pure program. The actual length N when compared with $\hat{N}$ gives a measure of maturity of the developer of the program. The following equation defines $\hat{N}$ and N.

$$\hat{N} = n1 \, Log_2 (n1) + Log_2 (n2)$$

Where n1 is the number of distincit operators and n2 is the number of distincit operands of a program.

$$N = N1 + N2$$

Where N1 is the number of total operators and N2 is the number of total operands.

Another aspect of programming which has been identified by Halstead is the language level (LL). Which measures the capability of the programmer in using the power of a given programming language. Higher value of LL reflects a better com-

mand on a given programming language. LL is calculated using the following formula:

$LL = \hat{L}^2 V$ where

$\hat{L} = \frac{2}{n1} * \frac{n2}{N2}$ represents estimated program level

and $V = N \text{Log}_2 (n)$ represents program volume

where Two tape drives with capacity 1600 BPI.

The third and possibly the most important concept introduced by Halstead is the intelligence content of a program denoted by IC. IC is calculated using the following formula

$$IC = LV$$

IC reflects 'how much' is expressed in a given program. It is language independent measure of the inherent content in a program. Therefore it nicely indicates the knowledge gained towards problem-solving capability.

In the next section we describe the data collection strategy used to facilitate the computation of above three important parameters of programs developed by students as part of their course work. It is our hypothesis that the variation in the above three parameters as the student progresses through the curriculumn starting from the freshman level to the senior level until his graduation reveals the effectiveness of the curriculum.

3. Data Collection

The College of Computer and Information Sciences here acquired a twin VAX-780 systems with the following configuration during 1984.

- Two CPU's.
- 16MB memory for each CPU.
- 64 direct connections for each CPU.
- Two tape drives with capacity 1600 BPI.
- Five disk drives: Two of them are removable (RA60) with capacity 205 MB and the others are fixed (RA81) with capacity 456 MB.
- A 1200 LPM parallel line printer.
- Two 300 LPM serial dot matrix line printer.
- Various relevant software packages.

Most of the programming exercises for the Computer Science Courses have been done using VAX-Pascal. The VAX operating system permits editing and stor-

age of Pascal programs by every student in his own protected directory. Students normally go through a lengthy debugging session for every program that they develop. The VAX system automatically gives new version numbers for the programs that are modified. Thus, the final version of a program has the highest version number.

We have collected the data utilizing the co-operation among the faculty of the Computer Science. As soon as the students submit the results of their assignments and projects, we were informed by the course instructors. We have developed a data-collection system which scans the directories of the students of a given course and automatically collects the latest versions of the most recent programs developed by them. Thus, the programs developed by every student of a given course are collected, collated and stored in our respective databases. Because the data-collection was done through co-operation and not through a mandate, the programs developed by a student as he progresses through the curriculum are not collected completely. Thus, sequential analysis of acquisition of programming skill by a specific student can not be done.

However, some bright students have been very co-operative. These students have sought the concurrence of the course instructors and have provided the programs developed by them in successive courses. Thus, we can state that, on an average, the progress in programming skills by the students of the department as they progress through the curriculum can be studied.

We have developed a software science laboratory consisting of various automatic analysis and database management tools. Databases for program samples were organized according to B.S. program structures. Within a course, samples were further classified to match a particular course implementation.

We would like to provide a brief information on program samples collected and analyzed during the investigation reported in this paper. Since all program samples analyzed were collected through the execution of the ten courses listed in the previous section over a period of six years, it seems logical that a brief description on implementation of each course under discussion will nicely expose the type and nature of program samples analyzed. The contents and implementation of the list of ten courses under consideration have a reasonable focus on theory and abstraction of Computer Science discipline as recommended by [1-6]. However, the design and implementation by students of concepts and techniques introduced during each course is heavily emphasized. Therefore, we set out to briefly discuss the areas covered by the programming-aspect of each course under investigation.

The first two courses on Computer Programming offer foundation in the general concepts of problem solving and provide thorough training in the Pascal programming language. Students are provided with a tutorial style presentation and are required to produce regular implementation of newly introduced concepts in the form of complete programs through assignments and projects. The program samples collected from both basic programming courses involve solution of problems in the area of numerical computation, text processing and searching and sorting of a given data.

Program samples collected from the Data-structures course revolve around problem solving and implementation of abstract data-type. Programs on all classical abstract data-types including Linear Structures, Trees and Graphs have been collected.

The course on File Processing Techniques focuses on studying the external data-structures necessary for implementing different file organizations. The program samples obtained from this course are based on implementation and problem-solving with various well-known file structures like: Indexed, Direct, ISAM, VSAM, Tree structure and external sorting.

Operating System courses demand a significant effort from students to implement techniques studied through various basic concepts on process management, scheduling, memory management I/O-handling and interrupt handling.

During first course on operating system each student is required to develop a small multiprogramming operating system on a simulated machine under VAX operating system. The average size of a student's project is about 2000 Pascal statements. All projects delivered by students are collected in our respective database for analysis.

During the second course on Operating System, students are given assignments to program concurrent processes, deadlock, and multitasking techniqes of operating system.

The Computer Network course aims at the exposure of principles of network programming on PC LAN. It involves programming on network filing systems, security and authentication and simulating data link layers functions.

The Compiler Design course requires students to implement lexical analysis, parser and code generation for a high-level language which is generally a subset of a Pascal-like language.

During the Software Engineering course, each student taking the course carries an individual project of his own choice. Such a project involves application of Software Engineering techniques from requirement analysis to implementation and testing. Students are exposed to the various software design methodologies. The projects here cover several areas of application including: Business, Systems, Graphics, CAI, etc. The average size of projects code collected here is about 3000 Pascal statements.

In the Graduation projects, which span over two consecutive semesters, students are required to work under the supervision of a faculty member. The graduation projects involve significant theoretical work, and implementation. Students produce relatively large programs covering a wide spectrum of applications both in interdisciplinary and Computer Science areas. Some collected project sizes are as large as 6,000 Pascal statements.

In summary, we have quite a varied collection of programs including: small, large sophisticated, crude, important, trivial, numerical and combinatorial. Our collection has the average program size of about 900 Pascal statements. The longest

program in the analysis has 7631 Pascal statements. Currently, about 15,000 Pascal program samples are available for analysis.

4. Analysis of Data

Using our software science laboratory, a set of given programs can be analyzed for a number of software metrics. These programs were used to calculate n1, n2, N1, N2, n, N, $\hat{N}$, LL and IC for the programs of each course separately. The averages of these parameters for every course is given in the Table. To ensure that spurious artifacts do not cloud our analysis of the curriculum effectiveness, we have applied statistical techniques of data-validation [31]. Then analysis has revealed that the data is of acceptable quality statistically.

Program purity

Looking at the column of values for $\hat{N}/N$, we find that the minimum value happens to be 0.51 and maximum happens to be 0.99. It should be observed that $\hat{N}$ is the expected size of corresponding pure program. A value of 0.99 means that the programs developed by the students are pure. The corresponding course is Computer Programming - II.

In the course Computer Programming - I, we introduce the student to algorithms for the solution of classical Computer Science problems. The students simply code the algorithm in Pascal using simple structures. The algorithms are of course developed by experts of Computer Science and hence can be considered to be pure. As the students are not familiar with the Pascal in the first course the ratio $\hat{N}/N$ is 0.92 and not 1. In the second course, Computer Programming - II, the students code algorithms given by experts. Since the students are now familiar with Pascal language, we find that the ratio $\hat{N}/N$ is 0.99.

We will consider the three courses, Data Structures, File Processing Technique and operating system-I as a single group for our discussion. In contrast with first two programming courses, Data-Structure course introduces new concepts. The programs in Data-Structures are not coding of faculty provided algorithms. The students have to design their own algorithms. The effect of student-designed algorithms is shown by the fact that the ratio $\hat{N}/N$ has decreased to 0.77. The other two courses build-up on the concepts learned in the Data-Structures course. Consequently even though the algorithms of File Processing Techniques and Operating System-I are student designed, experience and additional learning have increased the ratio to 0.86 for File Processing Technique and 0.83 for Operating System-I course. The course on Compiler Design introduces the theory of languages which does not depend upon the concepts of the earlier courses. Thus, eventhough the programs of the Compiler Design course are patterned after published programs, the ratio $\hat{N}/N$ further decreases to 0.69.

The rest of the four courses are project oriented. Because of the project orientation, in addition to algorithm design, the student performed system analysis also.

Table. Composite results (averages of software science metrics)

Name of Course	n1	n2	N1	N2	n	N	Ń	Ń/N	V	L	LL	IC
Computer Programming-I	22.50	44.33	258.83	180.40	66.82	439.23	511.24	0.92	2690.92	0.03	1.35	58.77
Computer Programming-II	22.75	59.70	411.55	285.10	82.45	696.65	686.56	0.99	4523.17	0.02	1.61	81.13
Data Structures	24.85	60.60	571.60	393.80	85.35	965.40	746.07	0.77	6267.99	0.01	1.10	76.28
File Processing Techniques	26.95	100.55	825.35	545.65	127.50	1371.00	1176.74	0.86	9687.12	0.01	1.73	131.04
Operating System-I	30.20	149.10	1289.45	835.70	179.30	2125.15	1772.13	0.83	16109.22	0.03	1.98	184.83
Software Engineering	27.60	178.80	1727.10	1269.00	206.40	2996.10	1586.34	0.53	23213.05	0.04	2.63	237.59
Compiler Design	31.0	119.15	1284.10	851.70	150.15	2135.80	1481.19	0.69	15498.52	0.01	1.64	140.33
Operating System-II	29.55	121.35	1435.00	1002.50	150.90	2437.50	1303.70	0.53	17534.00	0.02	1.73	146.96
Computer Network	29.85	153.30	1448.40	1033.60	183.15	2482.00	1267.97	0.51	18945.01	0.01	1.30	185.21
Graduation Project	41.10	454.40	6125.10	4589.70	495.50	10714.80	5947.49	0.56	97843.02	0.01	2.60	476.33

Thus, the programs of courses Software Engineering, Operating System-II, Computer Networks and Graduation Projects completely reflect the student's own effort. It is our opinion that the fact that the student is able to write programs which are about 50% pure implies the success of the curriculum. In conclusion from $\hat{N}/N$ values we find that the curriculum has helped the students to acquire programming skills.

Command over programming language

The parameter LL represents the command over the language Pascal. The changes in LL are similar to the changes in $\hat{N}/N$. We see that LL is 1.35 for Computer Programming-I and increases to 1.61 for the course Computer Programming-II. This decreases to 1.10 for Data-Structures and increases to 1.73 and 1.98 for File Processing Techniques and Operating System-I respectively.

Even though the programs of Compiler Design course are purer than those of Software Engineering, Operating System-II, Computer Networks, and Graduation Project, they are having a LL of 1.64 only in contrast to 2.63 of Software Engineering.

It is well-known in Software Science Theory that LL of input/output oriented program is less than that of computation intensive programs. The programs of Compiler Design course are table-driven and hence are in between input/output oriented and computation-oriented programs. The programs developed in Computer Networks course display Computer Network graphs on the VDU-Terminal. Thus, these programs are I/O-oriented. Accordingly, LL happens to be very low at 1.3.

Operating Systems-II is a project oriented course. Since an Operating System basically controls variety of peripherals, the LL is only 1.73. The LL of 1.98 for Operating System-I course is due to the fact that memory management algorithms are computation-oriented.

We can say that the course with the maximum impact on the quality of programming is Software Engineering. It is very heartening to see that LL for Software Engineering is highest at 2.63. The Graduation Project has LL of 2.6 and thus testifies that the students have indeed gained mastery over the given Programming Language. In summary, LL also indicates that the curriculum has achieved its objectives.

Intelligence content (IC)

The IC for programs of the first course are the lowest at 58.77. This increases to 237.59 for Software Engineering and 476.33 for Graduation Projects. This again establishes that students have indeed gained knowledge due to the curriculum.

Comparison with previous studies

Several studies have tested [32-34] software science metrics with impressive results. Most of the previous experiments were statistical studies. While comparing results of some of the major studies [33,34] conducted before, with our results, a number of interesting things becomes apparent. The averages of unique operators, unique operands and volumes are very small as compared to the averages of corres-

ponding metrics values of PL/1 in Elshoff [33] study. The average of the estimated program level in our study is significantly higher than that of Elshoff [33] study concerning PL/1. According to the software science metrics [32], lower values of n2 and N2 strongly influence the reduction of program difficulty and bugs.

Several interesting studies [32,34,35] have been performed to estimate relative level of various programming languages. These estimated levels are tabulated on a linear scale and are available in [32]. For example, it has been shown [32] that relative levels of Assembler (CDC) and PL/1 are 0.88 and 1.53 respectively. These findings confirm our common intuition of programming languages levels. In our study the Pascal Language Level (LL) varies from 1.10 to 2.63. The average value of LL for Pascal from overall experimentation came out 1.76. Interestingly, our investigation places Pascal level higher than PL/1 [32] and lower than level of APL [34] Languages. Since APL has been considered at a higher level of abstraction [34], it further validates our data-collection and measurements.

Metrics results and students performance

As stated earlier, the data-collection was not through a mandate. Therefore, it is hard to generalize any possible correlation between software science metrics results and students levels and their performance. Nevertheless, occasionally many interesting observations have been made in this regard for various students known to the author. Some students possessing good academic records and attending the second course on Computer Programming produced program having language level values between 6 to 10. These values are significantly higher for Pascal than the average value of LL reported earlier. Another similar group of good students taking the Software Engineering Course delivered the course projects having exceptionally higher values like 0.410, 0.572 and 0.596 of estimated program level. Program lengh and estimated length were almost equal for some program samples produced by few students (securing highest possible grade) in data-structures course indicating their maturity to develop pure programs.

For the courses taught by the author himself, the variations in software science metrics results were monitored during the progress of each single course. At one time, during execution of a data-structures course, program samples were collected and analyzed separately for all five assignments given to the students during the whole semester time. Interestingly, a progressive trend on the values of results of metrics LL and IC were observed. Similar observations were recorded on some other courses as well.

5. Summary and Conclusion

We have collected extensive data on the programs developed by students of B.S. degree in Computer Science over a number of years. Relevant software metrics like purity of programs, command over language, and intelligence content have been extracted from these data. From the analysis of the results obtained we find that the Computer Science curriculum, as a whole, has achieved its objectives. Our

experimentation clearly supports the relevant software metrics like purity of programs, command over language and intelligence content discussed here.

The study provides a firm basis to objectively evaluate effectiveness of the Computer Science curriculumn. The reported work may be extended on various avenues. We feel that mandatory collection of data would have helped us to follow the progress of each student. Such monitoring of student performance could provide early warning with respect to those students who do not successfully complete the program. It will be interesting to observe the patterns of the software metrics studied during the progress of a single course itself. Such a monitoring may help identify students requiring special assistance in that particular course to be of thrauptic value.

The software science laboratory developed provides environments for continuous experimentation and verification of software science theory. Further data collection and analysis by involving new batches of students may enhance the results reported in this paper. The proposed method and available data collection may have potential to support research in other areas. This may include experimentation to evaluate program design methods, design of programming languages and design and verification of new powerful software metrics.

Acknowledgement. The author would like to express his thanks to the referees for their constructive suggestions.

Reference

[1] Computing Curricula 1991. "A Report of the ACM/IEEE-CS Joint Curriculum Task Force". *CACM*, 34, No. 6 (June 1991).

[2] Curriculum '78. "A report of the ACM Curriculum Committee on Computer Science". *CACM*, 22, 3 (March 79).

[3] Denning, Peter J. *et al.* "Computing as a Discipline." *CACM*, 32, 1 (Jan 1989), 9-23.

[4] Grayson, L. P. "Education and America's Industrial Future." *IEEE Computer*, (June 1986), 10-16.

[5] Taylor, Booth *et al.* "Design Education in Computer Science and Engineering." *IEEE Computer*, (June 1986), 20-27.

[6] *Model Program in Computer Science and Engineering*. IEEE Computer Society Press, Order No. 932 (Dec 1983).

[7] Tartar, John *et al.* "1984 Snowbird Report: Future Issues in Computer Science." *IEEE Computer*, (May 1986), 101-105.

[8] Zelkowitz, Marvin V. *et al.* "Software Engineering Practices in the US and Japan." *IEEE Computer*, (May 1986), 57-66.

[9] Myers, W. "Computing Science Accredition Pro and Con." *IEEE Computer*, (June 1986), 47-49.

[10] Maier, J. H. "Computer Science Education in the People's Republic of China." *IEEE Computer*, (June 1986), 47-49.

[11] Rogers, Jean B. *et al.* "Preparing Precollege Teachers for the Computer Age." *CACM*, (March 1984), 195-200.

[12] Soloway, Elliot. "Learning to Program = Learning to Construct Mechanisms and Explanations." *CACM*, 29, No. 9 (Sept 1986), 850-858.

[13] McKeithen, K.B. *et al.* "Knowledge Organization and Skills Differences in Computer Programmers." *Cognitive Psychol.*, 13 (1981), 307-325.

[14] Soloway, E. and Ehrlich, K. "Emperical Studies of Programming Knowledge." *IEEE Trans.* Soft. Eng. SE-10, 5 (1984), 595-609.
[15] Soloway, E. and Iyengar, S. *Emperical Studies of Programmers*. New York: Ablex, 1986.
[16] Booth, T. and Miller, R.E. "Computer Science Program Accredition." *CACM*, 30, 5 (1987), 376-389.
[17] Gibbs, N.E. "The SEI Education Program." *CACM*, 32, No. 5 (1989), 594-607.
[18] Gries, D.; Walker, T. and Young, P. "The 1980 Snowbird Report: A Discipline Matures." *CACM*, 32, No. 3 (1989), 294-297.
[19] Butcher, D.F. and Muth, W.A. "Predicting Performance in an Introductory Computer Science Course." *CACM*, 28, No. 3 (March 1985), 263-268.
[20] Campbell, P.F. and McCabe, G.F. "Predicting the Success of Freshman in a Computer Science Major." *CACM*, 27, No. 11 (Nov 1984), 1108-1113.
[21] Alspaugh, C.A. "Identification of Some Components of Computer Programming Aptitude." *J. Res. Math. Educ.*, 3, No. 2 (March 1972), 89-90.
[22] Konvalina, J.; Stephens, L. and Williams, S. "Identifying Factors Influencing Computer Science Aptitude and Achievement." *AEDS J.*, 16, No. 2 (1983), 106-112.
[23] Mzlack, L.J. "Identifying Potential to Acquire Programming Skill." *CACM*, 23, No. 1 (June 1980), 14-17.
[24] Bateman, C.R. "Predicting Performance in Basic Computer Course." *Proc. of 5th Annual Meeting of the American Institute for Decision Sciences*, Allanta, GA: AIDS Press, 1973.
[25] Capstick, O.C.K.; Gordon, J.D. and Salvadori, A. "A Predicting Performance by University Students in Introductory Computing Courses." *SIGCSE Bull.*, 7, No. 3 (Sept 1975), 21-29.
[26] Fouler, O.G.C. and Glorefeld, L.W. "Predicting Aptitude in Introductory Computing: A Classification Model." *AEDS J.*, 14, No. 2 (1986), 96-109.
[27] Hosteller, T.R. "Predicting Student Success in an Introductory Programming Course." *Proc. of NECC5*, Silver Spring: IEEE Press, 1983.
[28] Howe, T.G. and Peterson, C.C. "Predicting Academic Success in Introduction to Computers." *AEDS J.*, 12, No. 4 (1979), 182-191.
[29] Halstead, M.H. *Elements of Software Science*. New York: Elsevier-North Holland, 1977.
[30] Ahmad, A. "An Emperical Analysis of Software Complexity Metrics." *Proceedings of 12th NCC*, King Saud University, Riyadh (21-24 Oct. 1990), 863-892.
[31] Morrison, D.F. *Multivariate Statistical Methods*. New York: McGraw-Hill, 1967.
[32] Shen, V.Y.; Conte, S.D. and Sunsmore, H.E. "Software Science Revisited: A Critical Analysis of the Theory and its Empirical Support." *IEEE TSE*, SE-9, No. 2 (March 1983), 155-165.
[33] Elshoff, J.L. "An Investigation into the Effects of the Counting Methods Used on Software Science Measurements." *SIGPLAN Notices*, 13, No. 2 (Feb 1978), 30-45.
[34] Konstam, A.H. and Wood, D.E. "Software Science Applied to APL." *IEEE TSE*, SF-11, No. 10 (Oct. 1985), 994-1000.
[35] Woodward, M.R. "The Application of Halstead Software Science Theory to Algol 68 Programs." *Software-Practices and Experiences*, 14, No. 3 (March 1984), 263-276.

طريقة منهجية لتقويم فعالية الخطط الدراسية لعلوم الحاسب

آفتاب أحمد

قسم علوم الحاسب، كلية علوم الحاسب والمعلومات، جامعة الملك سعود،
ص.ب. ٥١١٧٨، الرياض ١١٥٤٣، المملكة العربية السعودية

ملخص البحث. في خلال العقد الأخير قامت لجان من خبراء مختصين بتصميم الخطط الدراسية اللازمة للحصول على درجة البكالوريوس في علوم الحاسب، وقد استتبع ذلك قيام كثير من الخبراء بتحليل فاعلية المقررات الأولية في علوم الحاسب. وقد تبين من هذا التحليل أن أول مقرر في علوم الحاسب يعد ناجحًا. وهناك حاجة لوجود طرق وأدوات تحليلية للقياس الموضوعي لجودة مستوى مهارة البرمجة الذي يصل إليه الطلاب الدارسون لدرجة البكالوريوس في علوم الحاسب في خلال سنوات دراستهم. وقد قمنا بتطبيق النظريات التي ثبتت فعاليتها في مجال القياسات الكمية للبرمجيات على كم من البيانات التي تم تجمعها للتأكد من فعالية الخطة الدراسية الكاملة لعلوم الحاسب، وقد وجد أن هذه الخطة قد حققت أغراضها إلى حد كبير. وتعد الطريقة المقترحة والبيانات التي تم جمعها خطوة على طريق إلقاء المزيد من الضوء على التقويم الموضوعي للخطط الدراسية لعلوم الحاسب، كما سيكون أيضًا مفيدًا لمصممي الخطط الدراسية وكذا إدارات الجامعات والذين يسعون نحو «تقويم مردود» البرامج الدراسية لعلوم الحاسب، بالإضافة لذلك فإن هذا البحث يقدم وسيلة جيدة للتحقق من علم القياسات الكمية للبرمجيات.