J. King Saud Univ., Vol. 6, *Computer & Information Sci.*, pp. 1-16 (A.H 1414/1994)

Software Science Analysis of PROLOG

Aftab Ahmad
Department of Computer Science,
College of Computer and Information Sciences,
King Saud University, PO Pox 51178, Riyadh 11543, Saudi Arabia

(Received 25 April 1992; accepted for publication 17 November 1992)

Abstract. The theory of software science attempts formularization of software characteristics. It was developed by the late M. H. Halstead of Purdue University during the early 1970's. During the past years, large numbers of empirical and psychological studies were performed to validate and investigate capabilities of proposed software metrics. Most of these studies supplied evidence to support the theory. The on-going process of verification and refinement of theory initiated its application in wide areas including: estimating complexity of software maintenance and relating software errors to the theory.

Within the frame of these investigations, a great interest has been shown towards application of software science to programming languages for evaluation of their relative levels and testing of length equation. In this regard an extensive published work exists covering experimentation on various programming languages. However, all previous studies have been mainly concerned with application of software science to conventional 'Procedural Languages'. It obviously triggers motivations to explore applicability of software science to newly emerging 'Declarative Languages'.

The work in this paper reports on a pioneer investigation into the application of software science to a fifth generation declarative language like Prolog. Various tools and databases have been developed which provide on-line software metrics research environments and data-collection for declarative languages including Prolog. These tools were used to analyze large number of Prolog programs collected from published literature and university environments. Two fundamental counting rules and their possible interaction were studied for Prolog programs. The results from experimentation clearly indicate to the significant fact that software science can be successfully applied to Prolog. These results have nicely verified the length equation. The study concludes that Prolog has higher language level than all other programming languages previously studied. This prediction strongly confirms the common intuition on Prolog level relative to the other procedural languages.

Introduction

The rising cost of software development and maintenance have naturally aroused interest to quantify and analyze software complexity and quality. Maurice Halstead [1,2] developed a theory that gives objective measures of software complexity. Vari-

ous experimental and statistical studies [1-11] have shown very high correlations between the theory's predictions and actual program measures.

The most interesting studies performed were on several computer languages for evaluation of their relative levels. The language level metric expresses the power of a language relating a particular language facility to the mental effort required to use it. Higher language level means that production and maintenance of program is easier in that particular language. It also provides a basis to rank languages on linear scale. Language level metric may be used in selecting a language for new applications and in testing the potential power of a proposed language.

A series of studies have been performed by Halstead [1] and other researchers [5-11] to predict the language levels of various programming languages. These findings through software science metrics confirm our common intuition of programming language levels. It is worth mentioning that results of some studies like Woodward [6] on Algol 68 do not match with expectation. The Algol 68 language level predicted by software science was lower than that predicted for Assembler [11].

The era of fifth generation languages is emerging rapidly now. It is also being expected that these languages may find wider specialized applications during the near future. It nicely opens another avenue for software science experimentation and verification. Prolog is thought [12] to be significantly contributing towards future family of declarative and logical programming languages evolution. Therefore it seems interesting to experiment the applicability of software science to Prolog to assess its level with respect to previously predicted language levels.

Prolog is not just another programming language. It is significantly different from other computer languages in many respects. In Prolog, algorithm implementation differs from conventional languages. It is descriptive as well as prescriptive. Prolog encourages the programmer to describe situations and problems rather than the detailed means by which the problems are to be solved. This is of practical importance because the declarative aspects of programs are usually easier to understand than the procedural details. Prolog introduces the descriptive or declarative view of algorithm implementation. Intuitively, Prolog may be considered to be a higher level language than procedural languages. Therefore, the experimentation presented in this paper is an important investigation for producing evidence to support or contradict software science. If Prolog is a higher level language as intuitively suggested then accordingly software science metrics should place it above the other languages.

The work reported in this paper deals with application of all software science metrics on large number of Prolog programs. In section II, counting strategies are surveyed and counting variations are specified and various experiments have been proposed. Tools and data-collection on a program analyzed with its statistical analysis are presented in section III and results are discussed in section IV. Finally, the experiment is reviewed and experiences are summarized. The discussion on length and estimated length of Prolog programs with results are given in the appendix for interested readers. The composite results on rest of software science metrics are also presented here.

Counting Strategies

The field of software science [1,2] has undertaken significant research efforts in an attempt to provide useful metrics of various aspects of the programming process. Software science metrics assigns quantitative laws to the development and maintenance of computer software. Software science uses a set of primitive measures that may be derived after code is generated. These primitive measures are n1 (The number of distinct operators), n2 (The number of distinct operands), N1 (The total number of operators occurrences) and N2 (The total number of operands occurrences). The metrics n (Vocabulary) and N (Program Length) are given by (n1 + n2) and (N1 + N2) respectively.

Software science proposes a number of program impurities which reduces effectiveness of a program by increasing program length unnecessarily. The metric Estimated Length ($\hat{N}$) is an estimate of the length of a pure program. The $\hat{N}$ is given by $n1*\text{Log}_2\, n1 + n2*\text{Log}_2\, n2$.

The program volume V measures the number of bits required to specify a program in a given programming language. The volume V is defined as $N*\text{Log}_2\, n$.

When a programming language is rich in vocabulary of operators then it may allow more compact expression and produces shorter implementation of a given program.

Abstraction is considered as a major characteristic of software design to ensure its easy implementation. The estimated program level of abstraction $\hat{L}$ is given by (2/n1) * (n2/N2).

The program difficulty metric D2 is defined as the estimate of human difficulty in implementation of an algorithm in a given programming language. The metric D2 and Program Level have an inverse relationship. The metric D2 is simply given by $1/\hat{L}$.

Another interesting aspect of software quality which has been identified and proposed by software science is language level (λ) and is given by $\hat{L}2*V$. λ is an important aspect of programming process identified by software science metric to measure the inherent limitation imposed by the language. A higher value of language level reflects its power to decrease human difficulty for the implementation of a given algorithm. Language level metric estimates the level of abstraction provided by a programming language.

The software science effort metric E attempts to quantify the effort required to comprehend the existing implementation of an algorithm. The metric E represents the number of mental discrimination (decisions) that fluent, concentrating programmer should make to comprehend the implementation of algorithm in a given language. The effort metric E is given by $V/\hat{L}$.

The potential volume IC (or Intelligence Content) is metric that measures inherent content of a program and is given by $\hat{L}*V$.

Originally software science theory was meant for analyzing algorithms, where each symbol can be strictly classified as operator or operand. However, this classifi-

cation is not straightforward when algorithm is represented in computer programming languages. It provides room for a very large number of potential counting variations to be investigated for a particular programming language. However, there is a general consensus among researchers in the field now [14] that, as long as consistency is maintained, small details do not matter. Therefore, the following major counting strategies are being proposed in Prolog for data collection and analysis:

(a) Since Prolog is a declarative language, the declarative section represents a significant portion of programming effort. It does not appear reasonable to ignore declarative section of program completely.
(b) User-predicate symbols in Prolog may be viewed as functions or data variables.
(c) Generally a variable's scope is clause based. The occurrence of the same variable name in different clauses may be considered distinct operands.

Currently Prolog is available in many different versions. Most deviations among the different versions are of purely cosmetic nature. Among several Prolog dialects, the Endinburgh Syntax, also known as DEC-10 syntax [15], is most widespread.This syntax is well-documented by Clocksin and Melish in their popular reference [16]. Our work generally involves Edinburgh syntax with its enhancements like Turbo Prolog [17,18]. However Core Prolog [16] symbols are specially considered. It appears that Prolog allows enormously large variations for investigations. As a first-level attempt, we concentrate on some fundamental variations involving the declarative section and user-predicates only. Accordingly, four counting schemes are considered for experimentation and are summarized in the following Table.

Counting rules	Declarative section	User predicates
Expt # 1	Not considered	Operator
Expt # 2	Not considered	Operand
Expt # 3	Considered	Operator
Expt # 4	Considered	Operand

Clauses section is mainly considered for counting token. Goals section is ignored as in some Prolog implementation goals are provided at run-time of a program and not at compile-time.

Tools and Data Collection

Various intelligent automatic tools have been developed as part of this work. The integration of a strong, comprehensive tool support provides on-line Software Science Laboratory (SSL)' environments for analyzing and experimenting with Prolog programs.

The use of SSL involves sequence of tools and interaction of databases. SSL furnishes flexible capabilities for statically extracting and measuring a wide range of

program and language characteristics. It also provides facilities for analysis and manipulation of program measurements data. The tools offer the user ability to easily select from a range of capabilities provided by SSL.

The SSL static analysis capability can be subdivided into individually selectable capabilities offering these levels of analytic power. The lexical analysis step accepts as input the Prolog program source text and converts it into a list of lexical tokens. The syntactic analysis step requires the lexical tokens as input. This process constructs a parse tree representation of input Prolog program and symbol table. The static semantic analysis step builds upon the output of the first two static analyzers and produces a number of structures designed to represent and elucidate the functioning of the program. These structures facilitate extraction of any desired information on program or language attributes. Here syntax or structure of Prolog or programmer symbols such as: facts, predicates, clauses, questions, data-structures used (lists or trees), operators, expressions, variables and constants can be extracted and viewed. The static control and data flow structures can be also viewed and analyzed. At this level, computation of software science measures and metrics is straightforward.

SSL establishes and maintains a file system for creation, deletion, alter and manipulates files of source code modules and program measurements data sets. The file system furnishes input to the needed tools and captures the output data of those tools. The file system management is being done automatically and internal to SSL to keep user out of sight. The file system seems to be best thought of as relational databases. Therefore, relational database technology is applied for large bodies of data on program samples and analysis of programs. The interface of SSL encourages users to think about environments as an energetic, reasonably bright assistant, capable of answering questions and storing and retrieving important bodies of data. Through the user interface, one can easily attempt as many as possible counting strategies to view effects on all software science metrics. Various statistical analysis techniques are also available for data analysis. Data bases are maintained for ongoing collection of program samples and storage of analysis results. The automated laboratory integrates various tools and databases to provide environments for continuous research and experimentation with declarative languages like Prolog.

A good care has been taken in selection of program samples for analysis. Most of the samples analyzed have been collected from published literature like books on Prolog, programming language manuals and toolboxes including: [12,15-19]. Since the selected-programs are produced by researchers and expert professionals, it is assumed that these programs are pure in the software science sense. Similarly Prolog [17] demonstration programs and all programs provided in Prolog Toolbox [18] were considered for analysis. In fact the results presented in this paper for readers mainly contain programs from [17,18]. Within the Computer Science Department, various courses and projects involving Prolog programming are regularly offered. We have on-going data collection from students' assignments and projects. Similarly, programs produced through faculty research are also collected. Currently, about 1250

program samples are available. Most of the available program samples were collected from AI research groups at Cambridge and Edinburgh universities. Some program samples were obtained from Computer Science students and faculty at King Saud University. Out of these samples about 500 program samples are collected from students having an average program size of 400 lines of source code. These program samples nicely cover a wide spectrum of areas of applications including games, expert systems, graphics, string processing and simulation. Furthermore, the collected samples are produced by both expert and naive level programmers. The collected program samples are classified and maintained in various databases depending on their areas of applications and other characteristics.

Results and Discussion

All available program samples were analyzed using our tools and databases. For all four experiments, data on software science metrics and their statistical analysis is available. The summarized statistical results from overall experimentation will be presented and discussed in the later part of the paper. However, for the sake of clarity, we have chosen a group of 40 program samples for demonstration and discussion before mentioning results from overall experimentation. For these samples, data on various software science metrics is presented for all four experiments in Tables 1 to 4 respectively. The program samples 1-23 and 30-36 are taken from [17]. The rest of program samples are small expert systems for various applications developed by faculty and senior undergraduate students. The program samples taken from [17] are demonstrations for users, covering several characteristics of Prolog. We discuss results on software science metrics with a main emphasis on language level.

The averages, variance and standard deviation of language level and other metrics are provided in Tables 1-4 for 40 program samples analyzed. The Prolog language level observed through experimentation may be verified only by using frame of mean language level and their standard deviations obtained through Halstead's [1] and other experimental studies shown in Table 5. The ordering of language levels predicted by software science theory nicely confirms to the common intuition concerning levels of these programming languages.

The mean λ predicted by our all experiments taken together is 4.08. This places Prolog on the top of the Table 5. This result is in accordance with the logical expectation concerning Prolog. The original philosophy on language level as described by Halstead [1] and various critical reviews [14] on his theory may support the fact that Prolog should have higher-level than other programming languages. Prolog as a declarative language has a distinguished nature over other conventional procedural programming languages. It enhances the programming effort for implementation of a particular algorithm. It allows to implement a given algorithm in arbitrary ways.

The averages of λ have variations within four counting strategies used for analysis of programs. The significant effect of counting the declarative part of Prolog on λ is apparent. The consideration of user-predicates as operators or operands has

Table 1. Software science measurements of forty program samples based on experiment # 1

Prog.	n1	n2	N1	N2	n	N	$\hat{N}$	V	$\hat{L}$	λ	IC
1	15	30	109	70	45	179	205.81	983.04	0.06	3.21	56.17
2	10	6	18	14	16	32	48.73	128.00	0.09	0.94	10.97
3	17	23	71	39	40	110	173.53	585.41	0.07	2.82	40.62
4	21	33	114	76	54	190	258.70	1093.43	0.04	1.87	45.22
5	14	32	124	99	46	223	213.30	1231.75	0.05	2.63	56.88
6	33	79	358	190	112	548	664.46	3730.43	0.03	2.37	94.00
7	22	57	191	127	79	318	430.58	2004.60	0.04	3.34	81.79
8	10	18	46	33	28	79	108.28	379.78	0.11	4.52	41.43
9	18	26	118	65	44	183	197.27	999.08	0.04	1.97	44.40
10	9	9	61	44	18	105	57.06	437.84	0.05	0.90	19.90
11	14	17	50	34	31	84	122.79	416.15	0.07	2.12	29.73
12	19	41	145	87	60	232	300.37	1370.40	0.05	3.37	67.98
13	19	40	135	94	59	229	293.59	1347.13	0.04	2.70	60.34
14	17	43	129	57	60	186	302.82	1098.68	0.09	8.65	97.51
15	16	37	106	95	53	201	256.75	1151.31	0.05	2.73	56.05
16	12	25	105	71	37	176	159.12	916.86	0.06	3.16	53.81
17	16	42	139	101	58	240	290.48	1405.92	0.05	3.80	73.08
18	17	34	119	95	51	214	242.46	1213.90	0.04	2.15	51.11
19	5	40	69	63	45	132	224.49	724.92	0.25	46.76	184.11
20	20	29	104	61	49	165	227.32	926.43	0.05	2.09	44.04
21	15	25	80	44	40	124	174.70	659.92	0.08	3.79	49.99
22	24	94	372	271	118	643	726.17	4425.54	0.03	3.70	127.92
23	26	307	941	788	333	1729	2658.67	14487.95	0.03	13.01	434.19
24	19	41	132	79	60	211	300.37	1246.35	0.05	3.72	68.09
25	17	38	154	90	55	244	268.91	1410.65	0.05	3.48	70.07
26	20	43	187	129	63	316	319.77	1888.82	0.03	2.10	62.96
27	17	20	77	41	37	118	155.93	614.72	0.06	2.02	35.28
28	30	71	307	196	101	503	583.84	3349.08	0.02	1.95	80.88
29	16	18	60	37	34	97	139.06	493.48	0.06	1.82	30.01
30	9	6	28	13	15	41	44.04	160.18	0.10	1.69	16.43
31	11	17	61	40	28	101	107.54	485.54	0.08	2.90	37.52
32	9	16	35	24	25	59	92.53	273.99	0.15	6.01	40.59
33	13	16	47	25	29	72	112.11	349.77	0.10	3.39	34.44
34	15	13	39	24	28	63	106.71	302.86	0.07	1.58	21.87
35	12	17	40	24	29	64	112.51	310.91	0.12	4.33	36.70
36	15	19	45	30	34	75	139.31	381.56	0.08	2.72	32.22
37	12	13	53	25	25	78	91.13	362.22	0.09	2.72	31.39
38	18	30	93	58	48	151	222.27	843.33	0.06	2.79	48.47
39	60	145	727	420	205	1147	1395.50	8808.36	0.01	1.17	101.37
40	11	51	626	568	62	1194	327.35	7109.31	0.02	1.89	116.06
Average	17.33	41.53	160.38	111.03	58.85	271.4	321.41	1752.74	0.07	4.22	67.14
Variance	79.66	2532.0	38103.3	23774.5	3133.5	120694.5	200555.43	7464899.5	0.0081	52.05	4667.2
Std. dev.	8.93	50.32	195.2	154.19	55.98	347.41	447.83	2732.20	0.04	7.22	68.32

Experiment: 1 Correlation coefficient for N and $\hat{N}$ is 0.89199

Table 2. Software science measurements of forty program samples based on experiment # 2

Prog.	n1	n2	N1	N2	n	N	$\hat{N}$	V	$\hat{L}$	λ	IC
1	7	39	109	108	46	217	225.78	1198.61	0.10	12.76	123.67
2	10	7	18	17	17	35	52.87	143.06	0.08	0.97	11.78
3	13	27	69	52	40	121	176.49	643.95	0.08	4.11	51.44
4	18	36	114	85	54	199	261.18	1145.22	0.05	2.54	53.89
5	10	36	119	106	46	225	219.34	1242.80	0.07	5.73	84.42
6	20	92	358	244	112	602	686.61	4098.03	0.04	5.83	154.52
7	15	64	190	150	79	340	442.60	2143.29	0.06	6.94	121.93
8	9	20	46	43	29	891	14.97	432.36	0.10	4.62	44.69
9	12	32	118	96	44	214	203.02	1168.32	0.06	3.61	64.91
10	6	13	61	60	19	121	63.62	514.00	0.07	2.68	37.12
11	11	20	50	43	31	93	124.49	460.74	0.08	3.29	38.96
12	15	46	145	117	61	262	312.69	1553.85	0.05	4.27	81.46
13	14	45	142	113	59	255	300.44	1500.07	0.06	4.85	85.34
14	14	46	130	78	60	208	307.39	1228.63	0.08	8.72	103.51
15	15	38	123	105	53	228	258.02	1305.97	0.05	3.04	63.02
16	11	27	105	86	38	191	166.44	1002.35	0.06	3.27	57.22
17	11	48	167	127	59	294	306.13	1729.50	0.07	8.17	118.85
18	13	38	125	99	51	224	247.53	1270.62	0.06	4.43	75.03
19	5	41	87	72	46	159	231.27	878.25	0.23	45.57	200.04
20	12	37	104	92	49	196	235.77	1100.48	0.07	4.94	73.76
21	12	28	77	65	40	142	177.63	755.71	0.07	3.90	54.26
22	12	106	368	305	118	673	756.18	4632.02	0.06	15.54	268.30
23	26	307	1087	788	333	1875	2658.67	15711.33	0.03	14.11	470.85
24	14	46	132	108	60	240	307.39	1417.65	0.06	5.25	86.26
25	11	44	154	115	55	269	278.27	1555.19	0.07	7.53	108.19
26	14	49	187	159	63	346	328.42	2068.14	0.04	1.01	91.05
27	14	24	77	56	38	133	163.34	697.97	0.06	2.62	42.73
28	12	90	307	292	102	599	627.29	3996.78	0.05	10.55	205.31
29	9	25	60	58	34	118	144.63	600.32	0.10	5.51	57.50
30	6	10	28	26	16	54	48.73	216.00	0.13	3.55	27.69
31	9	19	60	47	28	107	109.24	514.39	0.09	4.15	46.21
32	7	18	31	28	25	59	94.71	273.99	0.18	9.24	50.32
33	11	18	42	30	29	72	113.11	349.77	0.11	4.16	38.16
34	14	15	41	31	29	72	111.91	349.77	0.07	1.67	24.18
35	11	19	40	30	30	70	118.76	343.48	0.12	4.55	39.55
36	12	22	45	41	34	86	141.13	437.52	0.09	3.50	39.13
37	10	15	53	34	25	87	91.82	404.02	0.09	3.15	35.65
38	14	34	93	76	48	169	226.28	943.86	0.06	3.86	60.32
39	25	160	690	510	185	1200	1287.60	9037.66	0.03	5.69	226.83
40	9	53	625	596	62	1221	332.11	7270.07	0.02	2.84	143.67
Average	12.32	46.35	164.43	132.20	58.67	296.63	326.35	1908.39	0.08	6.39	94.04
Variance	19.04	2634.13	43301.89	25990.31	2981.45	135659.89	195552.62	8432445.0	0.0015	50.8452	7133.53
Std. dev.	4.36	51.32	208.09	161.22	54.60	368.32	442.21	2903.870	0.04	7.13	84.46

Experiment: 2 Correlation coefficient for N and $\hat{N}$ is 0.90191

Table 3. Software science measurements of forty program samples based on experiment # 3

Prog.	n1	n2	N1	N2	n	N	$\hat{N}$	V	$\hat{L}$	λ	IC
1	19	31	129	87	50	216	234.29	1219.07	0.04	1.72	45.72
2	13	8	25	18	21	43	72.11	188.87	0.07	0.88	12.91
3	21	28	87	52	49	139	226.84	780.44	0.05	2.05	40.02
4	23	33	137	76	56	213	270.51	1236.97	0.04	1.76	46.70
5	16	32	130	99	48	229	224.00	1278.96	0.04	2.09	51.68
6	39	80	409	202	119	611	711.88	4212.73	0.02	1.74	85.56
7	27	59	214	137	86	351	475.46	2255.62	0.03	2.30	71.96
8	14	20	58	41	34	99	139.74	503.66	0.07	2.45	35.10
9	22	29	135	80	51	215	238.99	1219.57	0.03	1.32	40.19
10	13	11	76	56	24	132	86.16	605.22	0.03	0.55	18.29
11	18	18	64	43	36	107	150.12	553.18	0.05	1.20	25.73
12	24	44	162	100	68	262	350.25	1594.92	0.04	2.14	58.48
13	23	44	166	113	67	279	344.26	1692.44	0.03	1.94	57.30
14	21	45	159	78	66	237	339.37	1432.52	0.05	4.32	78.71
15	20	42	122	108	62	230	312.92	1369.47	0.04	2.07	53.26
16	14	25	114	71	39	185	169.40	977.80	0.05	2.47	49.19
17	23	53	174	138	76	312	407.62	1949.35	0.03	2.17	65.10
18	20	35	130	97	55	227	265.96	1312.37	0.04	1.71	47.35
19	10	41	84	71	51	155	252.88	879.23	0.12	11.73	101.54
20	24	30	125	71	54	196	257.25	1127.96	0.04	1.40	39.72
21	19	25	91	44	44	135	196.81	737.02	0.06	2.64	44.08
22	33	100	419	292	133	711	830.85	5016.31	0.02	2.16	104.12
23	26	307	941	788	333	1729	2658.67	14487.95	0.03	13.01	434.19
24	23	45	158	98	68	256	351.18	1558.39	0.04	2.48	62.22
25	22	39	173	98	61	271	304.24	1607.23	0.04	2.10	58.15
26	24	44	223	134	68	357	350.25	2173.22	0.03	1.63	59.47
27	22	21	100	44	43	144	190.35	781.38	0.04	1.47	33.90
28	34	72	373	237	106	610	617.21	4104.03	0.02	1.31	73.34
29	18	18	88	37	36	125	150.12	646.24	0.05	1.89	34.93
30	13	7	36	18	20	54	67.76	233.38	0.06	0.84	13.96
31	15	18	67	42	33	109	133.66	549.84	0.06	1.80	31.42
32	12	17	40	26	29	66	112.51	320.63	0.11	3.81	34.94
33	16	17	52	27	33	79	133.49	398.51	0.08	2.47	31.36
34	21	16	55	31	37	86	156.24	448.01	0.05	1.08	22.02
35	16	22	55	34	38	89	162.11	467.07	0.08	3.06	37.78
36	18	19	54	30	37	84	155.77	437.59	0.07	2.17	30.79
37	17	14	66	30	31	96	122.79	475.60	0.05	1.43	26.11
38	22	35	112	72	57	184	277.63	1073.25	0.04	2.10	47.43
39	69	154	751	455	223	1206	1540.57	9407.89	0.01	0.91	920.30
40	17	55	655	592	72	1247	387.46	7693.90	0.01	0.92	84.09
Average	21.27	43.15	176.32	119.98	64.41	296.2	354.43	1935.37	0.05	2.40	58.54
Variance	95.45	2520	39186.5	24257.8	3157.9	123558.2	202590.2	7711855.0	0.0005	5.8113	4150.87
Std. dev.	9.77	50.21	197.96	155.75	56.20	351.51	450.10	2777.02	0.02	2.41	64.43

Experiment: 3 Correlation coefficient for N and $\hat{N}$ is 0.89347

Table 4. Software science measurements of forty program samples based on experiment # 4

Prog.	n1	n2	N1	N2	n	N	$\hat{N}$	V	$\hat{L}$	λ	IC
1	11	40	129	134	51	263	250.93	1491.85	0.05	4.39	80.97
2	13	9	25	22	22	47	76.64	209.59	0.06	0.83	13.19
3	17	32	84	69	49	153	229.49	859.05	0.05	2.56	46.87
4	20	36	137	88	56	225	272.56	1306.65	0.04	2.19	53.45
5	12	36	122	110	48	232	229.14	1295.71	0.05	3.86	70.68
6	26	93	409	269	119	678	730.35	4674.69	0.03	3.31	124.32
7	20	66	212	167	86	379	485.37	2435.55	0.04	3.80	96.26
8	13	22	58	53	35	111	146.21	569.35	0.06	2.32	36.36
9	16	35	135	117	51	252	243.52	1429.45	0.04	2.00	53.45
10	10	15	76	76	25	152	91.82	705.87	0.04	1.10	27.86
11	15	21	64	55	36	119	150.84	615.22	0.05	1.59	31.32
12	20	49	162	135	69	297	361.56	1814.23	0.04	2.39	65.85
13	18	49	175	137	67	312	350.18	1892.62	0.04	2.99	75.21
14	18	48	161	102	66	263	343.14	1589.68	0.05	4.35	83.12
15	19	43	141	119	62	260	314.04	1548.09	0.04	2.24	58.88
16	13	27	114	88	40	202	176.49	1075.03	0.05	2.40	50.74
17	18	59	207	170	77	377	422.13	2362.58	0.04	3.51	91.11
18	16	39	136	105	55	241	270.13	1393.31	0.05	3.00	64.69
19	10	42	104	81	52	185	259.70	1054.58	0.10	11.34	109.36
20	16	38	125	110	54	235	263.42	1352.40	0.04	2.52	58.40
21	16	28	87	68	44	155	198.61	846.21	0.05	2.24	43.56
22	21	112	413	338	133	751	854.66	5298.52	0.03	5.28	167.21
23	26	307	1087	788	333	1875	2658.67	15711.33	0.03	14.11	470.85
24	18	50	158	132	68	290	357.25	1765.36	0.04	3.13	74.30
25	16	45	173	129	61	302	311.13	1791.08	0.04	3.41	78.10
26	18	50	223	170	68	393	357.25	2392.37	0.03	2.55	78.18
27	19	25	100	63	44	163	196.81	889.89	0.04	1.55	37.17
28	16	91	373	352	107	725	656.21	4887.56	0.03	5.10	157.94
29	11	25	88	65	36	153	154.15	791.00	0.07	3.87	55.31
30	10	11	36	35	21	71	71.27	311.85	0.06	1.23	19.60
31	13	20	65	51	33	116	134.54	585.15	0.06	2.13	35.30
32	10	19	34	32	29	66	113.93	320.63	0.12	4.52	38.07
33	14	19	45	34	33	79	134.01	398.51	0.08	2.54	31.81
34	20	18	59	40	38	99	161.50	519.54	0.04	1.05	23.38
35	15	24	55	42	39	97	168.64	512.68	0.08	2.98	39.06
36	15	22	54	44	37	98	156.71	510.53	0.07	2.27	34.04
37	15	16	66	41	31	107	122.60	530.10	0.05	1.44	27.58
38	18	39	112	94	57	206	281.19	1201.58	0.05	2.55	55.39
39	28	195	737	563	223	1300	1618.04	10141.17	0.02	6.21	250.89
40	15	57	653	622	72	1275	391.08	7866.65	0.01	1.17	96.1
Average	16.38	49.30	184.85	147.75	65.68	332.60	369.15	2173.68	0.05	3.30	77.65
Variance	18.54	2850.3	45237	27790.5	3189.91	143075.6	211515.2	9045671.0	0.004	6.3973	6130.08
Std. dev.	4.31	53.39	212.68	166.70	56.48	378.25	459.91	3007.6	0.02	2.53	78.29

Experiment: 4 Correlation coefficient for N and $\hat{N}$ is 0.90525

Table 5. Enhanced table of language levels of various programming languages obtained through previous experimental studies

Language	Level	Standard deviation
English	2.16	0.86
Prolog	2.03	0.59
APL	1.98	1.91
PL/1	1.53	0.96
Algol 58	1.21	0.86
Fortran	1.41	0.90
Pilot	0.92	0.66
Assembler (CDC)	0.88	0.65
IBM/370 BAL	0.43	0.83

also an effect on λ. In all four experiments, two program samples (19 and 23) have exhibited relatively large λ as compared to other samples. As it may be seen from tables that both samples have few unique operators (as compared to the total occurrences of operators) with very low operand redundancy. In fact, sample # 19 is referring only to one user-predicate repetitively throughout its clauses section. The program sample # 23 is a small example of classification expert-system given in [17]. It has almost similar situation of operators and operands as of sample # 19. Only two program samples have significant differences in λ under different counting methods. Examples are samples 1 and 22. Both samples have increased λ because of decrease in count of unique operators, when user-predicates are counted as operands. With these few exceptions, all program samples have lower variation from mean λ within all experiments. The standard deviations of λ in first two experiments is higher as compared to other experiments. However, in all experiments the standard deviations of λ are higher than standard deviations shown in Table 5 for other programming languages.

For the purpose of further refinements of λ, we eliminated all those program samples from analysis for each experiment having λ higher than 10. With this elimination, massive measurements using all available program samples were taken and summarized results are shown in Table 6. A significant shift on values of λ are apparent from Table 6. Here standard deviations of λ are significantly lower than those shown for 40 program samples. Both mean and standard deviation values are higher for the first two experiments than the rest of the experiments. It is apparent that counting declarative section and user-predicates as operators decreases language level. Also the values of mean λ for experiments 1, 2 and 4 are higher than language level predicted for English [1] as shown in Table 5. However, Halstead [1] has mentioned that English level should not be taken with any significance. Only experiment 2 has significant difference from other experiments for mean λ. Except experiment 3, the results of all other experiments place Prolog language level on top of Table 5. For experiment 3, the value of λ is lower than APL language level reported by De

Kerf [5] study and higher than APL level predicted by later study of Konstam and Wood [7]. However, Konstan and Wood [7] study has claimed various improvements over all previous APL-studies. The values of standard deviations of λ for all experiments are reasonably lower. Referring to Table 5, only experiment 3 has the lowest standard deviation of λ than reported for other programming languages. Also it has variance of λ significantly lower than variance calculated in our other experiments. Therefore, we strongly feel that language level calculated through experiment 3, appears to be intuitively appropriate.

Table 6. Composite results of overall experimentation on language level

Exp	Language level			Least square fit of λ on N	Correlation coefficient	
#	Average	var.	St. dev.	for N & λ	Slope	Intercept
1	2.79	1.7813	1.33	−.15051	−.00076	2.97115
2	4.12	3.5437	1.88	.12066	.00085	4.28832
3	2.03	0.3464	0.59	−.25750	−.00062	1.98798
4	2.71	1.5116	1.23	.22497	.00217	2.05225

The possible interaction between and N was analyzed. In Table 6 correlation coefficients of λ and N for all experiments are shown. Similarly, least square fit lines for λ on N given by $\lambda = a + bN$ were obtained. The values of a and b are tabulated in Table 6. From Table 6, it may be seen that correlation coefficients between length and λ are significantly lower and generally slopes of regression lines are shallow. It clearly hints that there is a weak tendency for λ to increase with length. It provides further evidence that Prolog has higher language level than other programming languages.

Our experimentation demonstrates that the unique operators count n1 tends to remain relatively stable with respect to the length of program samples analyzed for Prolog programs. Therefore, the analysis of results from our experimentation proposes that the average operands usage N2/n2 is the main contributor to the program difficulty D2. According to the software science metrics [1], the lower values of n2 and N2 strongly influence the reduction of program difficulty and program bugs. Pearson correlation coefficient between N2/n2 and D2 came out 0.9106 . The average of ratio N2/n2 from our overall experimentation is 2.95. This value is reasonably lower than the values of N2/n2 reported for other programming languages [1]. Interestingly, all four experiments have almost similar value of ratio N2/n2. This exposes very important internal property of Prolog, which remains invariant under all different counting strategies. The programming environments of Prolog have decreased the usage of number of variables and constants for implementation of the algorithm. The positive effect of philosophy of declarative languages has been nicely exposed by our experimentation.

A large portion of a programmer's time is spent in modifying existing programs to correct errors to meet new specifications. This activity, generally called 'program maintenance', requires a thorough understanding of existing code before additions or modifications can be made. The effort required to understand a given piece of software by a non-designer is often nontrivial. In fact, it is believed that the effort required to comprehend an existing program may be comparable or higher than that required for reprogramming from scratch. The importance of writing programs which are easy to comprehend has lead to the publication of many books on good programming style. It has been shown [1] through experimentation that software science metric E is a good predictor of comprehensibility. The λ has an interesting implication about the effort to understand an already developed program. Software science predicts that higher level languages reduce effort towards program understandability, maintenance and reusability. Furthermore, the equation of E shows that it is inversely proportional to the square of λ. Therefore, according to the software science metrics, square of λ indicates the efficiency of the corresponding language. From average values of λ and effort E computed from our experimentation, it becomes clear that Prolog needs less effort to understand a program than other procedural languages, as it has higher level than the that of other procedural languages computed before [1]. For example the relative efficiency for Prolog is more than twice as large as that for PL/I [1] and almost four times as large as for Fortran. This study has provided significant insights that benefits the Prolog programming community. It indicates on firm basis that Prolog programming environments may require relatively less effort for program development and maintenance.

Length and estimated length

The correlation coefficients of N and $\hat{N}$ for all experiments are shown in Tables 1-4 involving analysis of 40 programs. Similarly for overall measurements taken correlation coefficient of N and $\hat{N}$ together with slopes and intercepts of least square lines are shown in Table 7 for all experiments. Interestingly, all correlation coefficients are significantly higher indicating a very good correlation between N and $\hat{N}$ in all counting strategies. The average values of N and $\hat{N}$ are shown in Table 7 for overall measurements. Both values are reasonably closer within each experiment. However, the values of $\hat{N}$ and $\hat{N}$ have some differences when all experiments are considered. Generally, all data concerning N and $\hat{N}$ does not significantly indicate any preference for a one counting strategy over the other counting strategies. Also the results on N and $\hat{N}$ nicely agree with the findings for other programming languages (Algol, Fortran, and PL/I).

The composite results of all experiments for rest of software science metrics are shown in Table 8. The averages of n1, n2, and n are almost similar in four experiments. However, $\hat{N}$ and $\hat{N}_2$ have some variation over the counting strategies. Generally, these five measurements do not significantly distinguish counting method used.

The averages of volumes calculated in all experiments are higher than APL [7] and lower than PL/I [10]. The averages of $\bar{L}$ are similar to other programming languages.

Table 7. Analysis of interaction between N and $\hat{N}$

Exp #	Correlation coefficient of N and $\hat{N}$	$\hat{N}$	$\bar{N}$	Least square fit of N on $\bar{N}$	
				Slope	Intercept
1	0.90775	241.51	267.04	1.14983	9.34308
2	0.90194	238.42	244.42	1.08285	5.14594
3	0.92477	269.22	301.85	1.14407	15.45132
4	0.93660	303.19	318.96	1.10067	3.6461

Table 8. Averages of software science measurements on operators, operands, volume and program level

Exp #	n1	n2	N1	N2	n	V	L
1	7.65	35.08	145.14	96.38	52.73	1476.29	0.061
2	12.31	36.31	131.97	106.37	48.63	1426.25	0.073
3	21.70	36.39	162.30	106.92	58.43	1681.89	0.049
4	16.61	43.44	167.78	135.42	60.06	1899.13	0.052

Conclusion

The investigation has clearly indicated that software science theory can be applied to Prolog. The program length equation has been nicely verified. Averages of volume and program estimated level of Prolog are similar to other programming languages. Our study predicts that Prolog has higher language level than other procedural programming languages.

It appears that various counting strategies are possible for analysis of Prolog programs. However only two fundamental rules and their possible interaction have been investigated. These rules were expected to have serious impact on software science analysis of Prolog. Generally, software science measurements do not significantly prefer any counting strategy. However, language level varies under different counting rules. We would like to intuitively suggest that counting declarative section and user-predicates as operators appears to be most appropriate.

The study has provided a strong evidence that software science has capability to analyze Declarative Language' programs. The reported work may be extended on various avenues. In particular, psychological experimentation may be performed for data-collection on programmer performance to analyze a correlation between hours of effort and number of bugs. It is hoped that Prolog may yield some interesting results. Similarly the volume, potential volume and estimated program level may be further investigated.

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

References

[1] Halstead, M.H. *Elements of Software Science.* Elsevier Computer Science Library, Elsevier North-Holland Inc., 1977.

[2] Halstead, M.H. "Natural Laws Controlling Algorithm Structure." *ACM SIGPLAN Notices,* 7, No. 2 (Feb 1972), 19-26.

[3] Albrechrt, A.J. and Gaffney, H.E. Jr. "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation." *IEEE Trans. on Soft. Engg.*, SE-9, No. 6 (Nov. 1983), 639-648.

[4] Halstead, M.H. "Software Physics Comparison of a Sample Program in DSL ALPHA and Cobol." *IBM Research Report,* RJ 1460 (Oct 1974).

[5] Kerf-De, "API. and Halstead's Theory of Software Metrics." in *APL 81 Conf. Proc.* (Oct. 1981), 89-93.

[6] Woodward, M.R. "The Application of Hlastead Software Science Theory to Algol 68 Programs." *Software-Practices and Experiences.* 14, No. 3 (March 1984).

[7] Konstan, A.H. and Wood, D.E. "Software Science Applied to APL." *IEE Software Engineering,* SE-11, No. 10 (Oct 1985), 994-1000.

[8] Zweben, S.H. and Fung, K.C. "Exploring Software Science Relation in Cobol and APL." *Proc. IEEE COMPASAC '79* (Nov 1979), 702-709.

[9] Conte, S.D. "The Software Science Language Level Metric." *Dept. Comput. Sci.,* Purdue Univ., Rep. CSDTR-373 (1981).

[10] Zweben, Stuart H. and Halstead, M.H. "The Frequency Distribution of Operations in PL/1 Programs." *IEEE Trans. on Soft. Engg.*, SE-5, No. 2 (March 1979), 91-95.

[11] Smith, C.P. *"Practical Applications of Software Science."* IBM Santa Teresa Lab., Tech. Rep. 03.067 (June 1979).

[12] Walker, Adrian. *Knowledge System and Prolog.* Addison-Wesley Pub. Co. Inc., March 1987.

[13] Salt, N.F. "Defining Software Science Counting Strategies." *ACM SIGPLAN Notices,* 17(3) (1982), 58-67.

[14] Hammer, P.G. and Frewin, G.D. "M.H. Halstead's Software Science - A Critical Examination." in *Proc. 6th Int. Conf. Software Engg.*, Tokyo, Japan (Sept. 1982), 197-206.

[15] *DEC System-10 Prolog User's Manual.* Dept. of Artificial Intelligence, UNI of Edinburgh, Scotland.

[16] Clock Sin, W.F. and Mellisn, C.S. *Programming in Prolog.* Spring-Verlag Berlin, 1984.

[17] *Turbo Prolog.* Borland International, Inc., 1986.

[18] *Turbo Prolog Toolbox.* Borland International, Inc., 1987.

[19] Susan Garavaglia. *Prolog-Programming Techniques and Applications.* Harper & Row Pub., 1987.

تحليل علم البرمجيات للغة برولوج

آفتاب أحمد

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

ملخص البحث. أجريت أبحاث ودراسات عديدة سابقة مركزة أساسًا على تطبيق علم البرمجيات على اللغات الإجرائية التقليدية. وهذا بالطبع ولّد الرغبة في استكشاف تطبيق نظرية البرمجيات على اللغات الوصفية. ويقدم هذا البحث خطوة رائدة في إتجاه تطبيق علم البرمجيات على لغة وصفية من لغات الجيل الخامس مثل برولوج. وهناك العديد من الأدوات وقواعد البيانات التي تم تطويرها والتي تهيء بيئة بحثية مناسبة لقياسات البرمجيات وبها تضمنه من جمع بيانات للغات الوصفية بما فيها لغة برولوج مجمعة من المطبوعات المختلفة وكذلك من بيئة الجامعات. وقد تم استخدام قاعدتين أساسيتين للعدّ ودراسة إمكانية تفاعلها وذلك لبرامج برولوج. وقد أوضحت نتائج هذه التجارب أنه بالإمكان تطبيق علم البرمجيات بنجاح على لغة برولوج. وقد ساعدت هذه النتائج على التحقق من معادلة الطول. كما أوضحت هذه الدراسة أن لغة برولوج لها مستوى لغوي أعلى من لغات البرمجة السابق دراستها. وهذا يؤكد الاعتقاد السائد عن مستوى لغة البرولوج مقارنة بغيرها من اللغات الإجرائية.