which is assumed detectable and correctable by the operator, and is given as $(1-P)^6 + 6P(1-P)^5$.

III. Recommendations

The code described herein as an example includes zero and all one's as code words. An ideal code would not use these characters. Additionally, an optimum code would be one that contains the maximum number of characters and further investigation of this maximal set should uncover a subset in which some of the characters have the same error-correction properties as the numerals.

Programmer Variability

THOMAS E. DICKEY

Abstract—**Existing citations of the Sackman programming research are in error. This letter clarifies the scope of the research, and the conclusions which can be properly drawn from it.**

I. Scope of the Sackman Research

In 1966, Systems Development Corporation performed a research project for ARPA, whose intent was to determine if time-sharing systems were more effective than batch systems. An experiment was performed, using 12 experienced programmers to code and debug two programs, one each on a time-shared system and on a batch system. The batch system was simulated by imposing a two-hour turn-around on the debugging runs. All *coding*, in each case was done off-line, and on a "simulated" batch system). Thus, 24 measurements were used, in four cells. The results of this experiment were published in [1] and [3]. A summary was published in the *Communications of the ACM* [2]. The ACM paper is the most often cited; however, it does not include the experimental data. Only a summary of the data are presented in [2], together with a somewhat misleading commentary. This discussion focuses on the original data.

As shown in the Table I (reproduced from [1] and [3]), the experimental group of 12 programmers was divided into two groups each of six programmers who programmed one program each on time sharing and batch. On inspection, several aspects of the data are obvious.

1) Three of the programmers programmed in SCAMP, the machine language of the system which was used for the experiment, while the other nine programmed in JOVIAL Time Sharing (JTS), a dialect of ALGOL.

2) One of the nine programmers had no prior experience with time sharing. From the discussion in [1], one finds that subject 2 learned JTS in order to accomplish the experiment. Of the set of 12 programmers, only 8 were familiar with JTS.

3) The groups of programmers were imbalanced in their time-sharing experience; Group II had 4.6 times as much.

The programmer selection problems listed in the preceding were discussed in [1] and [2]; Sackman and Grant felt that these factors were not sufficiently important to redesign the experiment. Thus, for the purpose of analysis, the ALGOL and machine language programming tests were lumped together. In actuality, the experiment size is not sufficient to determine the relative efficiency of the machine language programming; their inclusion is misleading. If the original set of eight JTS programmers is studied alone, the experiment is somewhat more balanced. A *balanced* decomposition of the debugging data (after the experiment design) shows that Group I debugs 2.12 times *slower* than Group II, primarily due to subject 4, who is inexperienced (3 years). Subject 4 debugs 5 times slower than six of the remaining seven programmers. He is partly offset by programmer 2, the best and most experienced (11 years) programmer of the eight. Although lack of space precludes an error analysis, the time-shared system was 2.53 times faster than the batch, primarily due to the batch turn-around of two hours.

In [2], the mean and standard deviation for the debugging times are presented to qualify the results of the experiment. However, rather than present the total of the experimental data, a summary is presented showing the *extremes* of debugging time for all programmers. Thus, subject 7 required 170 hours to program the "algebra" program in a batch environment, in machine language. Subject 3 required 6 hours to program the same problem in JTS (ALGOL) in a time-shared environment. The ratio of these two (28.3 = 170/6) was presented as an example of the "range of individual differences in programming performance" [2]. Similar ratios were presented for each component of debugging time, coding time, CPU time, program size and run time.

These ratios are misleading because they encompass all differences due to

1) differences between time sharing and batch systems;
2) differences between the performance of JTS programmers and machine language programmers;
3) differences between the programmers on the basis of their prior knowledge of the time-sharing system.

After accounting for the differences in classes, only a range of 5:1 can be attributed to programmer variability. The casual researcher, in encountering Sackman's paper, seizes on the 28:1 figure primarily to support arguments to the effect that programmer variability is "orders of magnitude" larger than effects due to language and system differences.

II. Further Citation of the Sackman Paper

The original Sackman paper has become a body of literature, which is extensively cited. At the NATO conference on software engineering in October 1968 [4], one of the panelists mentioned the range-of-performance figures in [2]. From this authoritative reference, the 28:1 figures made their way into the trade literature [5], from which point they were extensively copied, as in [6]. In retrospect, it appears that this single source, by means of different paths, is responsible for a large percentage of the common stock of "knowledge" that programming productivity is totally unpredictable. For example, in Yourdon's book [7], the performance range is quoted in support of the premise that programming productivity varies over at least an order of magnitude, even with "highly experienced" subjects. Weinberg [8] is somewhat more cautious, and points out that Sackman did not take into account the quality of the test programs, but simply whether or not the programs successfully executed the test cases. Other examples, though not exhaustive, are found in [9]–[13].

The most recent case of this misinterpretation of the Sackman experiment occurs in [14], where Curtis relates that Sackman observed 25-30:1 differences in performance among programmers.

References

[1] E. E. Grant and H. Sackman, "An exploratory investigation of programmer performance under on-line and off-line conditions," System Development Corporation, Santa Monica, CA, SP-2581, Sept. 2, 1966.

[2] H. Sackman, W. J. Erickson, and E. E. Grant, "Exploratory and experimental studies comparing on-line and off-line programming performance," *Commun. Assoc. Comput. Mach.*, vol. 11, no. 1, pp. 3-11, Jan. 1968.

[3] H. Sackman, *Man-Computer Problem Solving.* Auerbach Publishers, 1970, pp. 38-47.

[4] P. Naur and B. Randell, Eds., "Software engineering," Report on a conference sponsored by the NATO Science Committee (Garmisch, Germany), Oct. 7-11, 1968. (Brussels, Belgium: Scientific Affairs Division, NATO, 231 pp., 1969, p. 83.)

[5] J. L. Ogdin, "The Mongolian hordes versus superprogrammer," *Infosystems*, vol. 19, no. 2, pp. 20-23, Dec. 1972.

[6] "Issues in programming management," *EDP Analyzer*, vol. 12, no. 4, 14 pp., Apr. 1974.

[7] E. Yourdon, *Techniques of Program Structure and Design.* Englewood Cliffs, NJ: Prentice-Hall, 1975.

[8] G. M. Weinberg, *The Psychology of Computer Programming.* New York: Van Nostrand, 1971. p. 29.

[9] B. W. Boehm, "Software and its impact: A quantitative assessment," *Datamation*, pp. 48-59, May 1973.

[10] F. P. Brooks, Jr., *The Mythical Man-Month.* Reading, MA: Addison-Wesley, 1975, p. 88.

[11] D. Van Tassel, *Program Design, Efficiency, Debugging, and Testing.* Englewood Cliffs, NJ: Prentice-Hall, 1978, p. 98.

[12] E. Yourdon and L. L. Constantine, *Structured Design.* Englewood Cliffs, NJ: Prentice-Hall, 1979, p. 292.

[13] B. Curtis, S. B. Sheppard, P. Milliman, M. A. Borst, and T. Love, "Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 96-104, 1979.

[14] B. Curtis, "Measurement and experimentation in software engineering," *Proc. IEEE*, vol. 68, pp. 1144-1157, Sept. 1980.

Manuscript received October 30, 1980.

The author is with Westinghouse Research and Development Center, 1310 Beulah Road, Pittsburgh, PA.

TABLE I
EXPERIMENTAL DATA MATRIX

	Criterion Measures for Debugging Performance				Additional Measures for Programmer Performance						Control Variables				
	Debug Hours Algebra	Debug Hours Maze	CPU Time Algebra (sec)	CPU Time Maze (sec)	Code Hours Algebra	Code Hours Maze	Program Size Algebra	Program Size Maze	Run Time Algebra (sec)	Run Time Maze (sec)	TSS Experience (months)	Gen. Prog. Experience (years)	Prog. Lang. (SCAMP = 0, JTS = 1)	BPKT 1 (Programming Knowledge)	BPKT 2 (Coding Skill)
Group I On/Off	On	Off	On	Off											
S 1	15	4.5	749	82	11	20	4433	1538	3.2	6.8	2	2	J	32	8
S 2	29	20.5	1781	240	76	23	3215	1371	2.0	3.7	0	7	J	51	14
S 3	6	7.0	678	57	24	4	1050	651	3.5	2.8	9	11	J	35	12
S 4	57	26.0	1228	273	58	6	4957	3227	3.5	.6	4	3	J	38	6
S 5	15	8.0	1434	401	66	5	4042	1890	7.9	3.9	6	5	J	45	13
S 6	85	8.0	1727	91	111	50	2460	1286	2.4	4.2	2	11	S	47	5
Group II On/Off	Off	On	Off	On											
S 7	170	12.5	3075	257	88	16	2140	1501	3.5	2.4	30	4	S	47	12
S 8	31	2.0	370	541	19	2	3550	2188	5.0	5.4	3	2	J	50	11
S 9	20	3.0	403	115	12	2	1186	1000	3.0	2.3	18	10	S	43	10
S 10	27	2.0	527	269	60	16	6137	3287	6.6	8.0	12	9	J	25	5
S 11	23	3.5	416	141	7	2	2286	933	2.2	2.0	10	8	J	46	12
S 12	30	1.0	652	50	24	10	2690	1348	1.6	1.8	32	4	J	32	4
Mean I	34.50	12.33	1266.16	190.66	57.66	18.00	3359.50	1660.50	3.75	3.66	3.83	6.50	0.83	40.50	9.00
SD I	30.52	8.72	473.38	136.42	36.22	17.67	1437.63	867.72	2.12	2.01	3.25	3.88	0.40	9.81	3.57
Mean II	50.16	4.00	907.16	228.83	35.00	8.00	2998.16	1709.50	3.65	3.65	17.50	6.16	0.66	41.33	9.66
SD II	58.85	4.25	1067.06	174.64	32.01	6.92	1719.26	894.19	1.85	2.51	11.51	3.25	0.51	7.44	3.82
F-test	0.33	4.42	0.56	0.17	1.31	1.66	0.15	0.00	0.00	0.00	7.82	0.02	0.38	.02	.09
Prob.	-	.10	-	-	-	-	-	-	-	-	.05	-	-	-	-