PROGRAMMER'S EXPERIENCE AS A CLASSIFICATION VARIABLE IN STATISTICAL ANALYSIS OF SOFTWARE DATA

BY
JORGE LUIS ROMEU

IIT RESEARCH INSTITUTE, TURIN RD. NORTH
P. O. Box 180, ROME, NY 13440

ABSTRACT

This paper addresses the problem of looking at the intuitive complexity evaluation of a software job by experienced programmers, and of using it in conjunction with some statistical techniques. This analysis may help to identify which variables, if any, follow the qualitative pattern defined by the programmer's complexity criteria. Following this approach, a set of (complexity) characterization variables may be obtained and be used to build models for the control/forecast of the productivity and cost of the software development process.

1.0 STATEMENT OF THE PROBLEM

The production of a large software system is a complex activity involving different factors. Characterizing the complexity of this activity through the selection of a relevant set of end-product variables, such as, size, number of operators and operands, cyclomatic complexity, function points, is crucial for software performance evaluation, for estimation of future costs and schedules, and for trade-offs studies.

In turn, these end-product characterization variables are themselves determined by multiple factors (Fig. 1), not all of them known or quantifiable. The end-product variables may correlate highly with the abstract concept of complexity, but remain poor forecasting indicators.

However, experienced programmers do have an intuitive qualitative appreciation of the programming complexity of a software task, for complexity definitely exists and varies from one task to the next. The information stemming from their practical experience can be very valuable if it is properly used.

This paper addresses the problem of looking at the intuitive complexity evaluation of a software task by experienced programmers, and of using it in conjunction with some statistical techniques. This investigation may help to identify which variables, if any, follow the qualitative pattern defined by the programmer's complexity criteria. Since variability in complexity is readily present in software, good predictor variables should vary, reflecting the same pattern.

Also addressed here is the analysis of the relationships between the precision of program specification and the complexity of the programming task itself, as perceived by the programmer. Program specification is a controllable activity. If it is statistically verified that the subjective assessment of quality of specifications do affect the degree of subjective assessment of program complexity in a given direction, it will provide the software engineer with an additional elements to deal with the complexity issues.

We believe this paper provides two contributions. First, it provides an analysis and some statistical results that may be of interest to software researchers and practitioners, and which can easily be followed up with additional software data analyses for confirmation in other environments. Second, it illustrates the advantages in the application of some statistical techniques developed and used in other scientific areas to the field of software engineering, by way of a real life example.

2.0 METHODS

2.1 The Data

The statistical results presented here were obtained in the analysis of the NASA/SEL dataset[1] at the Data and Analysis Center for

1 The NASA/SEL dataset consists of software development data collected by the Software Engineering Laboratory (SEL) from NASA/SEL

projects. It contains over 45,000 records, mainly from component status and run analysis reports. The remainder is comment information and change, resource summary and component summary reports.

Software (DACS). These analyses are conducted under the DACS Baseline effort which is aimed at providing software engineers with some production baselines. For additional information about the NASA/SEL dataset and the Baselines task see (SEL81), (ROM83a) and (ROME84).

The quantitative variables used in the analyses were size, effort, and productivity. The three programmer's subjective evaluation classes for module complexity were easy, medium and hard. The three programmer's subjective evaluation classes for precision of the specifications were very precise, precise and imprecise. All of these variables were obtained from the Component Summary File (CSF). The subjective variables were the result of each programmer evaluating the degree of programming difficulty and the degree of specification clarity of each module developed at the start (stage NEW), as well as at the end (stage COMPLETE), of their programming task. Specific units of measurement are irrelevant to the statistical result of the tests performed.

Modules under analysis were classified into 12 function groups according to the functions they perform within their corresponding programs, as shown in Table 1. It was assumed that different module functions induce different software characteristics. This post stratification scheme provides a greater homogeneity in the data, within groups, and more easily identifies the software variable differences, between groups. Not all module function groups appearing in Table 1 had enough data to support the analyses reported here.

The two qualitative variables, subjective evaluation of module complexity and of precision of the specifications were classified into three increasing degrees of difficulty. To avoid the problem of a "grey zone" in the process of variable selection, the quantitative variables of size, effort and productivity were compared only when their corresponding module complexity evaluation was either 'hard' or 'easy' (that is, all 'medium' complexity modules were discarded for this analysis).

Since subjective or qualitative variables reflect an order of preference, as perceived through the programmer's experience, they are assumed to be given in an ordinal measurement scale level. The quantitative variables (for example, size) are also assumed to be given in a ordinal measurement scale level. The reasons for this interpretation are discussed in detail in (ROM83b).

2.2 The Statistical Methods

Several nonparametric techniques were employed in these analyses with two distinct objectives: (1) The comparison and selection of variables that reflect software complexity, and (2) the study of the relationships between complexity and precision of the specifications.

TABLE 1: NASA/SEL ENCODING DICTIONARY

CODE	MODULE/ COMPONENT NAME	FUNCTION
1	INCLUDE	INCLUDE STATEMENTS
2	CONTROL	CONTROL STATEMENTS (JCL, OVERLAY)
3	SYSTEM	SYSTEM STATEMENTS (ALC)
4	GESS	GRAPHICS STATEMENTS (GESS)
5	DATA	DATA STATEMENTS
7	CDR	FORTRAN CONTROL/DRIVER MODULE
8	C COMP	FORTRAN CONTROL/COMPUTATIONAL STATEMENTS
9	DTRANS	FORTRAN DATA TRANSFER MODULE
10	IO	FORTRAN INPUT/OUTPUT MODULE
17	IOCDR	FORTRAN CONTROL/DRIVER MODULE WITH I/O
18	IOCCOMP	FORTRAN CONTROL/COMPUTATIONAL MODULE W
19	IODTRANS	FORTRAN DATA TRANSFER MODULE WITH I/O

(1) Selection of (end-product) complexity characterization variables

Assuming the programmer's assessment of the coding task at completion of module development (stage COMPLETE) is accurate, there will be a statistically significant difference between 'easy' and 'hard' subgroups within software characterization variables (within size, effort, etc.). The null hypothesis of no difference was tested against two alternatives:

(a) 'easy' and 'hard' subgroups differ in location (that is, that one group is statistically larger that the other). This was done through the Wilcoxon-Mann-Whitney rank-sum test (LEHM75)

(b) 'easy' and 'hard' subgroups differ in dispersion (that is, that one group is more variable than the other). This was done through the Siegel-Tukey test for dispersion (SIEG60)

(2) Relationships between programmer's subjective assessment of module complexity and his subjective assessment of precision of the specifications

Verifying that an imprecise specification induces a complex programming task (or vice versa) may prove very useful. Under such conditions, to dedicate more effort to develop specification standards, design languages, requirements analysis methods, etc., may well pay its way handsomely. The null hypothesis of no association was tested against the ordered alternative that the lower the perceived precision of the module's specification, the higher the perceived complexity level of the coding task. This hypothesis was tested through ordered alternatives contingency table procedures (LEHM75). This hypothesis was also tested both at the initiation (NEW stage) and at completion (COMPLETE stage) of module development. The statistical results at both of these (NEW and COMPLETE development stages) were also compared within each module function.

3.0 RESULTS

Results of the first type of the analyses are reported in Table 2. The variables size, effort and productivity are compared at stage COMPLETE by Module Function. Observe that across module functions, the variable 'size' (with the exception made of module function 4, Graphic Statements) exhibits a consistent, significant difference between complexity groups. Easier modules yield smaller sizes and complex modules tend to be larger. This is consistent with the strong correlation that has been traditionally found between lines of code and other complexity indicators. The converse may also be true; that is, that some longer modules may be classified erroneously as complex.

Observe that for variable 'effort', with the exception of module functions 4 and 9, 'easy' modules indicate smaller development efforts. This is also consistent with traditional results and may serve to validate the accuracy of the programmer's subjective complexity evaluations, and hence, the other statistical results presented here.

Differences between complexity groups in the third variable, 'productivity' do not show a definite pattern. In some cases, differences in productivity are nonsignificant, or the direction of the significant differences, when detected, is not consistent. In some cases, easy is smaller; in others it is larger; and in yet others, the difference is not in location but in variability. These results agree with traditionally poor results when correlating sizes with productivity.

The association between programmer's assessment of complexity and precision of the specifications is reported in Table 3. This association has been analyzed both by module function and by development stage. As above, the behavior of the variables by module function and stage is studied in order to contrast the programmer's expectancies with the project's realizations. Observe how, with the exception of module functions 7, 8 and 9, the hypothesis of association between increasing module complexity and decreasing precision of the specification is highly significant. Also, contrast how at stage NEW (at initiation of the coding of a module) this test result is usually nonsignificant, while it becomes significant at the completion stage.

Finally, a practical interpretation of Table 3, which reports on the tests for association between precision of specifications and perceived complexity, for ordered alternatives, will be provided through an example. For Module Function 19 (data transfer with I/O) the association between perceived complexity and precision of specifications was highly significant at stage Complete (column 3). This led to the acceptance of the alternative hypothesis that a decrease in precision of the specifications is highly associated with an increase in programming task complexity. The last three columns of Table 3, report the actual order in which the complexity classes resulted after projects were ordered, following a precision of specifications criteria and then counted and averaged by perceived complexity classes.

Notice, in the case of Module Function 19, the projects having the highest rating of specification precision also had the lowest average rank of perceived complexity (easy = Rank 1; medium = Rank 2; and Hard = Rank 3).

TABLE 2: STATISTICAL RELATION BETWEEN PERCIEVED COMPLEXITY AND DEVELOPMENT CHARACTERISTICS

MODULE FUNCTIONS	SIZE	EFFORT	PRODUCTIVITY
MF4 Graphics Statements	NO RELATION	NO RELATION	EASY IS SMALLER (*)
MF7 Control/Driver	EASY IS SMALLER (**)	EASY IS SMALLER (**)	NO RELATION
MF8 Control/ Computational	EASY IS SMALLER (*)	EASY IS SMALLER (***)	EASY IS LARGER (**)
MF9 Data Transfer	EASY IS SMALLER (*)	HARD IS MORE DISPERSED (***)	HARD IS MORE DISPERSED (**)
MF10 Input/Output	EASY IS SMALLER (***)	EASY IS SMALLER (***)	EASY IS LARGER (**)
MF17 Control/Driver with I/O	EASY IS SMALLER (***)	EASY IS SMALLER (***)	EASY IS LARGER (***)
MF18 Control/ Computational with I/O	EASY IS SMALLER (**)	EASY IS SMALLER (**)	EASY IS MORE DISPERSED (***)
MF19 Data Transfer with I/O	EASY IS SMALLER (**)	EASY IS SMALLER (*)	EASY IS MORE DISPERSED (**)

Legend

*	Test is significant at level $\alpha = 0.05$
**	Test is significant at level $\alpha = 0.01$
***	Test is significant at level $\alpha = 0.001$

TABLE 3: PERCEIVED COMPLEXITY VS. PRECISION OF SPECIFICATIONS: ORDERED ALTERNATIVES

MODULE FUNCTION	DEVELOPMENT STAGE	ASSOCIATION TEST OUTCOME	AVERAGE RANK ORDER OUTCOME EASY	MEDIUM	HARD
MF19 Data Transfer with I/O	N	non Significant	N/A	N/A	N/A
	C	Highly Significant	1	2	3
MF18 Control/ Computational with I/O	N	non Significant	N/A	N/A	N/A
	C	Significant	1	2	3
MF17 Control/Driver with I/O	N	Significant	1	2	3
	C	Highly Significant	2	1	3
MF10 Input/Output	N	Highly Significant	1	2	3
	C	Highly Significant	1	2	3
MF9 Data/Transfer	N	Significant	1	2	3
	C	non Significant	N/A	N/A	N/A
MF8 Control/ Computational	N	non Significant	N/A	N/A	N/A
	C	non Significant	N/A	N/A	N/A
MF7 Control/Driver	N	non Significant	N/A	N/A	N/A
	C	non Significant	N/A	N/A	N/A
MF5 Data Statements	N	non Significant	N/A	N/A	N/A
	C	Highly Significant	1	2	2
MF4 Graphic Statements	N	non Significant	N/A	N/A	N/A
	C	Highly Significant	3	1	2

<u>Legend</u>

N/A	non applicable	
non Significant	Test is non significant at level	$\alpha = 0.05$
Significant	Test is significant at level	$\alpha = 0.05$
Very Significant	Test is significant at level	$\alpha = 0.01$
Highly Significant	Test is significant at level	$\alpha = 0.001$

* Average rank order is computed from the ranks of all observations within each treatment. It is an indicator of the actual (ordered) outcome of the test.

4.0 CONCLUSIONS

The dataset used in these analyses represents very specific types of application and environment. It remains to be seen if the same results hold for other applications/environments. In addition, some of the data from the CSF file were machine generated or estimated (for example, module function classification was produced by an algorithm).

However, the analysis presented in this paper is not dependent on specific data characteristics. The statistical results were obtained with rank-based nonparametric procedures, whose robust properties are well known. Finally, several partial results are in close agreement with previous, well-established experiences and/or research performed (see section 3.0, paragraphs 1, 2 and 3). Hence, statistical results reported here may be cautiously interpreted and further validated with additional data, but their conclusions are regarded as valid.

Overall conclusions:

(1) Size and effort are strong contenders for characterizing software complexity, but are insufficient by themselves.

(2) Productivity is a poor characterization variable for reflecting the complexity of software.

(3) A similar analysis approach can be followed with additional software variables (operators, operands, function points, cyclomatic complexity) for the screening and selection of a set of complexity characterization variables.

(4) There is a strong indication that the less precise the program specifications, the greater the program or module complexity (or vice versa), as perceived by the programmer. The programmer is, in the final analysis, the one who deals with the task complexity and consumes the coding effort.

5.0 ACKNOWLEDGEMENTS

Thanks are due to the NASA/SEL organization (especially to Mr. Frank McGarry) for furnishing the data to perform these analyses; to Mr. Robert Vienneau (IITRI/DACS), for his programming; to DACS Program Manager, Thomas Robbins, for his encouragement and support; and to many colleagues at IITRI-Rome for their helpful interaction.

6.0 REFERENCES

(1) (LEHM75) Lehman, E. L. <u>Nonparametric Statistical Methods Based on Ranks</u>. Holden-Dey, San Francisco. 1975.

(2) (SEL-81) Weiss, D. et al. <u>Software Engineering Laboratory</u>. SEL-81-002. September 1981.

(3) (SIEG60) Siegel, S. and J. Tukey. "A Nonparametric Sum of Ranks Procedure for Relative Spread in Unpaired Samples." JASA, Vol. 55, pp 429-444. (1960).

(4) (ROME83a) Romeu, J. L. "An Approach to Software Baseline Generation." Proceedings of the 1983 NASA/Goddard Software Engineering Workshop, Washington, DC.

(5) (ROME83b) Romeu, J. L. and S. Gloss-Soler. "Some Measurement Problems Detected in the Analysis of Software Productivity Data and their Statistical Consequences." Proceedings of the COMPSAC'83 Conference. Chicago, Ill.

(6) (ROME84) Romeu, J. L. Baselines Effort Progress Reports No. 1, 2 and 3. DACS (Draft) Reports August 1983, April and December 1984.