Toward a Detailed Classification Scheme for Software Maintenance Activities

Evelyn J. Barry
Graduate School of Industrial Administration, Carnegie Mellon University
email: eb48@andrew.cmu.edu. phone: (412) 268-3681, fax: (412) 268-7064
Chris F. Kemerer
Katz Graduate School of Business, University of Pittsburgh
Sandra A. Slaughter
Graduate School of Industrial Administration, Carnegie Mellon University

ABSTRACT

Concern for Y2K compliance emphasizes the need for understanding and improved management of software maintenance activities. Relatively little empirical research has examined the type and extent of activities taking place during software maintenance. Our research represents a first attempt in developing a detailed taxonomy that describes the type and distribution of activities within software maintenance. We illustrate our taxonomy using maintenance data from an actual application system.

INTRODUCTION

Software maintenance activities span a system's productive life and consume a major portion of the total life cycle costs of a system. However, what is actually done to systems in maintenance is sometimes a mystery to many organizations. Thus, software maintenance remains an opaque activity that is expensive and difficult to manage. With this study, we intend to open the "black box" of software maintenance by developing a detailed classification scheme for software maintenance activities.

BACKGROUND

A few researchers have attempted to identify the major types of maintenance work. One of the first typologies for maintenance activities was proposed by Swanson (1976). Swanson's classification of maintenance types into *adaptive*, *perfective* and *corrective* later evolved into the IEEE standard for software maintenance. Pressman (1992, pp. 664-665) describes four different types of maintenance: *corrective*, *adaptive*, *preventive* and *perfective*. The empirical work that has been done tends to report most (at least 50%) of the maintenance effort as perfective (Lientz and Swanson, 1980, p. 68). However, neither classification scheme describes in detail the kinds of work done *within* perfective maintenance.

PROPOSED TAXONOMY

To develop a more detailed taxonomy, we begin by examining the reasons for software maintenance. The traditional view of software maintenance was to 'fix' program source code. As Lehman and Belady (1985, p. 12) pointed out, fixing software differs from repairing manufactured products. Software does not wear out or deteriorate with age. Repairing manufactured goods restores products to their original state. Problems with software actually reveal flaws in the original source code or specifications. Like Swanson (1976), we classify this type of maintenance as *corrective*. We formally define *corrective maintenance* as software maintenance activity intended to adjust software to correct some type of erroneous output. Output may include user interfaces, or output internal to the system.

A second reason for software maintenance is to adapt software so that it will comply with a changing technological environment. The technological environment may include hardware and software such as operating systems and compilers. Quite often some adjustments must be made to software so that systems will continue to function after these changes occur. This maintenance effort adds no new functionality. Maintenance work done for the purpose of adapting to a changing technological environment is classified as *adaptive*. We formally define *adaptive maintenance* as software maintenance activity intended to adjust software to comply with changes in the technological environment. Such adjustments are required in order for continued operation. *Adaptive maintenance* includes version upgrades, conversions, recompiles, and the re-assembly, and restructuring of code.

A third reason for software maintenance activity is to expand software capabilities and features. Practitioners and researchers often refer to this as perfective maintenance or enhancement. Software enhancement increases application functionality by adding new source code or through extensive modification to current source code. Software features added during the enhancement process go beyond the work needed for adaptive maintenance. Enhancements give the system users additional capabilities not previously available. We classify maintenance work done for the purpose of expanding and improving the functionality of an existing software system as *enhancement*. We formally define *enhancement* as software maintenance activity done to increase the functionality of a software system.

Our objective in this study is to focus on enhancement activities because the proportion of effort devoted to enhancement has frequently been assessed as the largest in the software life cycle. Enhancements to source code are done for a number of different reasons. Building upon the classification scheme for maintenance proposed by the SEL at the University of Maryland (Rombach, Ulery, and Valett, 1992), we have elaborated six sub-types of enhancement activities (Table 1). These sub-types correspond to the major kinds of software functions (data handling, control flow, initialization, user interface, computation, module interface and initialization). Within each of these sub-classifications, we have further characterized the enhancements in terms of additions, changes or deletions. Our next step is to demonstrate this classification scheme empirically.

EMPIRICAL EVALUATION

Our research site is a large mid-Western retailer that has many legacy systems under maintenance. We examined maintenance activities for one of the retailer's oldest systems: the financial sales reporting system (FSR). Over the course of this system's 20-year history, the maintainers kept a detailed log of every modification made by recording date, purpose and type of change. We coded each event in the system's change history according to our proposed classification scheme (Kemerer and Slaughter, 1999).

Counts of each enhancement by sub-type were accumulated by month for the 20 years of change history spanned by our study. In addition, the growth pattern for the number of enhancement tasks was plotted for each of the six main sub-types. We observe that 78% of all modifications were enhancements, and that most enhancements in this system involve changes or additions to data handling and control flow logic (Table 2). The cumulative plot (Figure 1) shows that enhancements did not accumulate at a steady rate over the life of this application. Rather, modifications grew dramatically in the later years, spurred by the significant addition of new modules and changes to the system beginning in 1986.

Interestingly, in this system, most enhancements involved changes or additions of source code. Relatively few enhancements actually deleted code. Practitioners often believe that old systems accumulate large amounts of "spaghetti code", much of which is not executed. Longitudinal analyses of change event histories and module complexity would help to substantiate this belief. Our next step in this study is to develop methodologies for analyzing and comparing patterns of maintenance activity across applications. A deeper understanding of the types, distribution and patterns of maintenance activities can open up the "black box" of software maintenance and help improve software change management.

SELECTED REFERENCES

Kemerer, C.F. and Slaughter, S.A., "An Empirical Approach to Studying Software Evolution", IEEE Transactions on Software Engineering, 1999, *forthcoming*.

Lehman, M.M. and Belady, L.A., Program Evolution: Processes of Software Change, Academic Press, London, 1985.

Lientz, B.P. and Swanson, E.B., Software Maintenance Management, Addison-Wesley Publishing Co., Reading, MA, 1980.

Pressman, R.S., Software Engineering: A Practitioner's Approach, 3rd ed., McGraw-Hill, New York, NY, 1992.

Rombach, H.D., Ulery, B., and Valett, J., "Toward Full Life Cycle Control: Adding Maintenance Measurement to the SEL," Journal of Systems Software, vol. 18, 1992, pp. 125-138.

Swanson, E.B., "The Dimensions of Software Maintenance," Proceedings of the 2nd IEEE International Conference on Software Engineering, 1976, pp. 492-497.

IMPLICATIONS AND CONCLUSIONS

Sub-Category	*Description*
Data Handling	Modifications to definition or formats of data structures, segments, parameters, databases or files
Control flow	Modification to the control flow logic of the program
User Interface	Modifications to screens, messages, reports or any other type of output sent to the users
Computation	Modification to formulas and algorithms
Module Interface	Modifications to the code creating information for another module or another system to access
Initialization	Modification in the code setting constants or initial values for variables

Table 1: Descriptions of Sub-Categories for Enhancement Activities

	Add		*Change*		*Delete*		*Total*	
	Count	*Percent*	*Count*	*Percent*	*Count*	*Percent*	*Count*	*Percent*
Data handling	82	14	157	27	29	5	268	47
Control flow	43	7	163	28	21	4	227	40
User Interface	19	3	24	4	0	0	43	7
Computation	8	1	14	2	5	1	27	6
Module Interface	2	0	4	1	0	0	6	1
Initialization	2	0	1	0	0	0	3	1
All Enhancements	156	27	363	63	55	10	574	100

Table 2: Counts of Enhancements to FSR Note: percentages may not sum to 100% due to rounding

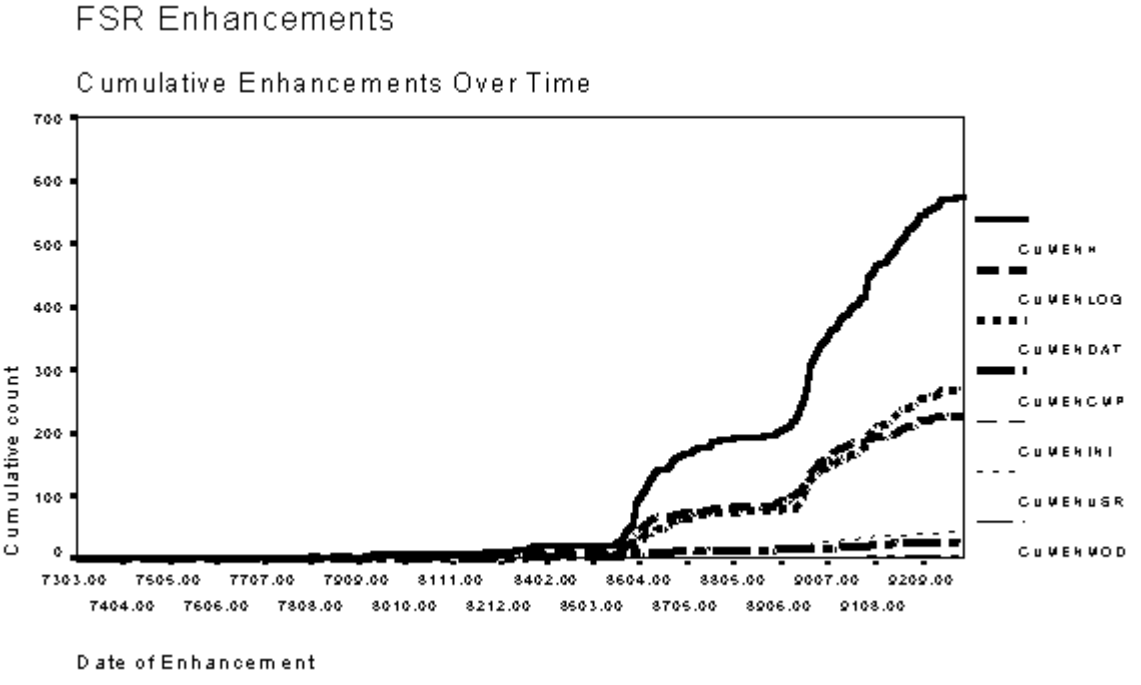

Figure 1: Cumulative Enhancements for Financial Sales Reporting System (FSR)