# AN EMPIRICAL ANALYSIS OF SOFTWARE EVOLUTION PROFILES AND OUTCOMES

**Evelyn Barry**
**Sandra Slaughter**
Graduate School of Industrial Administration
Carnegie Mellon University
U.S.A.

**Chris F. Kemerer**
Katz School of Business
University of Pittsburgh
U.S.A.

## Abstract

If anything good can be said to have come from the Year 2000 systems problem, it is that it has created a heightened awareness of how-long lived most software is and how extensively software maintenance efforts span a system's productive life. One of the most important maintenance processes deserving study is *software evolution*,described as the dynamic behavior, growth and change of software systems over their productive lives. Due to the difficulties in obtaining longitudinal data for empirical analysis of software evolution, little scientific knowledge is available about how software systems evolve and the consequences of different evolutionary patterns. Our objectives in this program of research are to identify software evolution profiles (or patterns of maintenance activities) and to link these evolution profiles to important maintenance outcomes such as software errors and costs. Traditionally, researchers classified software maintenance activities in three major categories: *corrective*, *adaptive* and *enhancement*. We expand this classification scheme to identify six enhancement subcategories related to major software functions (data handling, control flow, initialization, user interface, computation and module interface). After building software lifecycle maintenance profiles described by these classifications, we link these profiles to maintenance outcomes by developing a two-stage model. The first stage links software complexity to variables such as application size, maintenance team size, and the influence of purchased (vs. in-house) software applications; The second stage links software errors to software maintenance profiles, application usage and software complexity. We conducted an empirical study of the software lifecycle maintenance activities in a large Midwestern retailer, examining 21 batch applications over 31 months. Results indicate that lifecycle maintenance profiles do affect software outcomes. A significant portion of software errors can be explained by enhancements to control flow during the current time period, and enhancements to initializations and computations from the prior period. Our investigation of the effects of software lifecycle maintenance profiles, or evolutionary patterns, demonstrates the importance of increased understanding of software evolution. Continued work will include research on assessing the effects of maintenance profiles on maintenance costs as well as software errors. In addition, detailed examinations of a greater number of maintenance types and maintenance team characteristics, and incorporation of software stability measures, are expected to be completed.

**Keywords:** Software evolution, software maintenance, software enhancement, software complexity, software management

# 1. INTRODUCTION

If anything good can be said to have come of the Year 2000 systems problem, it is that it has created a heightened awareness of not only how dependent society is on computer systems, but, more specifically, how long-lived most software is. As Zvegintzov (1998) has observed, software systems are often in place for decades. As a consequence, there is greater cognizance of how much of the cost of computer systems is due to their maintenance over time. Software maintenance activities span a system's productive life and can account for as much as 80% of the total effort expended on a software system (Robson et al. 1991). One of the most important maintenance processes deserving study is *software evolution*. Software evolution is "the dynamic behavior of programming systems as they are maintained and enhanced over their life times" (Belady and Lehman 1976). Software evolution is of increasing importance as systems in organizations become longer-lived. In addition to the difficulties in collecting empirical data on software systems, empirical studies of software evolution face particular challenges due to the longitudinal nature of evolution. As a result, there is relatively little scientific knowledge about how software systems evolve and the consequences of different evolution patterns. This highlights the need for careful research as the basis for improving management of software maintenance processes.

In this program of research, we seek to understand how software systems evolve in organizations. The specific objectives of the current study are to identify *software evolution profiles* (or patterns of maintenance activities) and to link these evolution profiles to important *maintenance outcomes* such as software errors and costs.

# 2. BACKGROUND AND THEORETICAL MODEL

## 2.1 Software Evolution Profiles

Our first objective in this study is to identify profiles of software evolution activities. We must first characterize the types of activities transpiring during a system's productive life. A few researchers have identified the major types of maintenance work. Swanson (1976) proposed one of the first typologies for maintenance activities. Swanson's classification of maintenance types into *corrective*, *adaptive*, and *perfective* later was adopted by the IEEE as the standard for software maintenance (IEEE 1993).[1] Some empirical research has analyzed software modification activities to discover unique features found in software maintenance. For example, Gefen and Schneberger (1996) found that adaptive maintenance occurred at a fairly steady pace throughout software lifecycles. Other studies of maintenance activities report that most modifications to systems involve perfective maintenance (also known as *enhancements*). Typically, these studies report that organizations devote more than 50% of maintenance effort to enhancements (Dekleva and Zvegintzov 1991; Nosek and Palvia 1990). However, these maintenance classification schemes have not detailed the kinds of activities done *within* the major maintenance categories. Given the significant role previously observed for enhancement activity, in this study, we concentrate our efforts on a detailed analysis of this category.

We developed subcategories of enhancements by building upon the classification scheme for maintenance proposed by the Software Engineering Lab (SEL) at the University of Maryland (Rombach, Ulery and Valett 1992). We elaborated six subtypes of enhancement activities. These subtypes correspond to the major kinds of software functions (data handling, control flow, initialization, user interface, computation, and module interface). Enhancements modifying data handling activities pertain to data formats, record segments, databases or files, and establishment of parameters. Enhancements to control flow include references to changes in logic and program structure. Initialization enhancements refer to any source code modifications establishing constants or initial data values. User interface enhancements include modifications of human-computer interfaces, e.g., screens, reports, error and warning messages. Computation enhancements include modifications for equations and functions. Enhancement of module interfaces refers to changes in communication links between modules and/or submodules. In this study we refer to the history of these software enhancements as the *maintenance profile* for an application.

[1] An additional type of activity—*preventive* maintenance—has been identified by Pressman (1992, pp. 664-665).

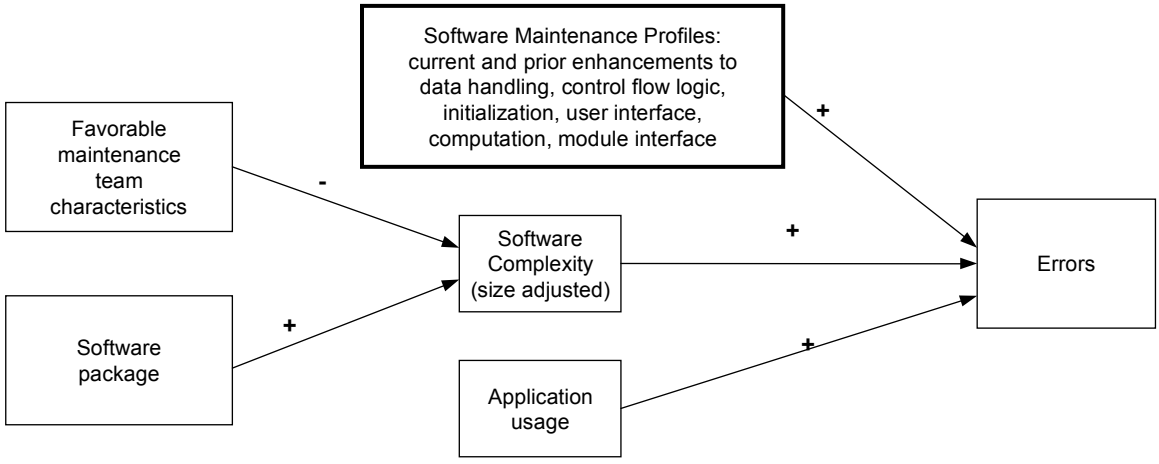


**Figure 1. Two-stage Model of Software Maintenance Profiles and Outcomes**

## 2.2 Software Evolution Profiles and Maintenance Outcomes

Our second objective in this study is to identify the consequences of different enhancement profiles for maintenance outcomes such as costs and errors. Our initial focus is on software errors, particularly software failures in a batch environment.[2] When software fails, managers frequently chase obscure, yet pressing, problems with few diagnostics describing the cause of those problems (Swanson and Beath 1989). If software managers could predict the frequency of production problems they will face, they could become more proactive, and their ability to plan and manage their work would be greatly enhanced. To explain the occurrence of software failures in batch processing, we identify software evolution activities likely to predict the frequency of abnormal terminations (abends) in batch processing. We empirically test the effect of each of our subcategories of software enhancements, but also include other explanatory control variables likely to lead to software failures. Our conceptual model is presented in two stages (see Figure 1). The first stage of the model examines variables contributing to software complexity. The final stage predicts the frequency of batch abends (errors), using maintenance profiles, and controlling for the effects of software complexity and application usage.

After classifying all enhancements according to functionality, we examined types of enhancements done in the current and previous time periods. While the expected effect of all changes is to increase the number of errors, all else being equal, the effects of some enhancement changes may not become immediately apparent in production environments. Some enhancements are more difficult to implement. Some are not tested thoroughly before implementation. New and modified source code for these enhancements may not be executed for several time periods after implementation. Therefore, we use a lag term for software enhancement categories to recognize that enhancement affects outcomes beyond their implementation time period.

We include control variables for amount of application usage and software complexity. Application usage is an obvious control variable, since the more times the software is executed, the greater the likely number of observed errors. In addition, software complexity is frequently included in analyses of the relationships of software characteristics and software maintenance efforts. A number of studies confirm that software complexity contributes significantly to maintenance costs and errors (e.g., Banker et al. 1997; Kemerer 1995). Having specified software complexity as an antecedent to software errors, a reasonable managerial question is then to ask what factors contribute to software complexity? We hypothesize that maintenance team characteristics affect software complexity. We control for application size and differentiate between purchased software packages and applications developed in-house. Larger applications are expected to be negatively associated with software complexity as it is

[2]In later stages of this study, we plan to examine software failures in an online environment and also maintenance costs.

**Table 1. Conceptual (Operational) Variables**

| *Conceptual (operational) variable* | *Description* |
|---|---|
| Software maintenance profile | Counts of enhancements in each of the six functional subcategories for each period |
| Data handling(DATAHNDL) | |
| Control flow (LOGIC) | |
| Initialization (INITIAL) | |
| User interface (USERINTR) | |
| Computation (COMPUTE) | |
| Module Interface (MODINTR) | |
| Errors (ABNDBTCH) | Count of run-time abends in batch processing each period |
| Software complexity (N1FP) | Application total of number unique operators normalized by application function points |
| Application usage (TRNSBTCH) | Count of transactions processed each period |
| Application size (APPLFP) | Total number of function points in each application |
| Package indicator (PACKAGE) | Dummy variable to indicate purchased software |
| Maintenance team characteristics (PGMRCNT) | Number of programmers working on application each period |

operationalized here, since the level of complexity is also adjusted for size in the denominator (Gill and Kemerer 1991). All else being equal, the use of a software package is expected to be positively associated with software complexity, given that the focus of this study is software maintenance, and the maintenance work is being done by people who were not the original developers.

The next stage in our investigation is to operationalize our conceptual model and conduct an empirical analysis to test the significance of these factors. This empirical analysis examines batch applications only. Online applications will be analyzed later. Conceptual and operational variables for this study are explained in Table 1.

## 3. EMPIRICAL ANALYSIS

Due to the longitudinal nature of lifecycle maintenance activities, empirical studies of the phenomenon require longitudinal data. Given the practical difficulties in collection of longitudinal data, it is of note here that we have been able to draw from a 20 year history of maintenance activities for a portfolio of 21 batch applications.

### 3.1 Research Site

Our research site is a Midwestern retailer with a large portfolio of legacy systems under maintenance. Over the course of the application portfolio's 20 year history, software maintainers kept a detailed log of every modification made to each module within each application by recording date, purpose, type of change and programmer responsible. We coded each event in the system's change history according to our classification scheme for enhancements (Kemerer and Slaughter 1999). Monthly counts by enhancement subtype were accumulated for 20 years of change history. Application and module characteristics were combined with a 31 month history of abends, and number of transactions processed. This provided us with a full data set covering 21 batch applications for 31 months.

### 3.2 Empirical Evaluation

Two equations were developed from our conceptual model (variable names can be found in Table 1):

$$[1]\ ABNDBTCH = \alpha_1 + \beta_1 TRNSBTCH + \beta_2 N1FP + \beta_3 DATAHNDL_t + \beta_4 LOGIC_t + \beta_5 INITIAL_t + \beta_6 USERINTR_t + \beta_7 COMPUTE_t + \beta_8 MODINTR_t + \beta_9 DATAHNDL_{t-1} + \beta_{10} LOGIC_{t-1} + \beta_{11} INITIAL_{t-1} + \beta_{12} USERINTR_{t-1} + \beta_{13} COMPUTE_{t-1} + \beta_{14} MODINTR_{t-1}.$$

$$[2]\ N1FP = \alpha_2 + \gamma_1 APPLFP + \gamma_2 PGMRCNT + \gamma_3 PACKAGE.$$

We estimated these equations using nested recursive regressions. After correcting for autocorrelation of the residuals, separate ordinary least squares regressions were run on each equation. Resulting parameter estimates are consistent and efficient. (Greene 1990, p. 596-597). Regression analysis for equation [1] indicates five significant variables can explain batch abends. The other variables were insignificant in the regression. The resulting regression was significant with F = 109.62 (p = 0.000) and R-square of 0.47. Specific results of the regression after correction for autocorrelation of residuals are listed in Table 2 (see discussion section below).

We regressed software complexity, N1FP, against the explanatory variables in equation [2]. The resulting regression was significant with F = 74.02 (p = 0.000) and R-square of 0.27. Specific results of the regression after correction for autocorrelation of residuals are listed in Table 3.

**Table 2. Linear Regression of Batch Abends (Equation 1)**

| | *Coefficient* | *Std. Err.* | *t* | *P > \|t\|* |
|---|---|---|---|---|
| TRNSBTCH | .0154 | .0007 | 21.285 | 0.000 |
| N1/FP | 1.0399 | .5120 | 2.031 | 0.043 |
| LOGIC (t) | .1053 | .0561 | 2.679 | 0.008 |
| COMPUTE (t-1) | 3.0625 | 1.1031 | 2.776 | 0.006 |
| INITIAL (t-1) | 1.6003 | .5771 | 2.773 | 0.006 |
| All other variables were insignificant | | | | |

**Table 3. Linear Regression of Software Complexity (Equation 2)**

| | *Coefficient* | *Std. Err.* | *t* | *P > \|t\|* |
|---|---|---|---|---|
| APPLFP | -.0008 | .0001 | -13.288 | 0.000 |
| PACKAGE | .0346 | .0139 | 2.484 | 0.013 |
| PGMRCNT | -.0023 | .0004 | -5.908 | 0.000 |

## 4. DISCUSSION AND CONCLUSIONS

These preliminary results indicate that lifecycle maintenance profiles affect software maintenance outcomes. We demonstrate that a significant portion of abends can be explained by enhancement activity profiles. Current enhancements to control flow logic are a significant factor. Of the 5,608 enhancements recorded in our 31 month panel data, 2,931 or 55% of them are enhancements to control flow. Data handling enhancements represented the next largest subclassification with 1,809 change events. There is a significant correlation (0.29) between data handling enhancements and those for control flow. To avoid multicollinearity problems, we excluded data handling enhancements from our model. Prior month enhancements to initializations and computations also contribute significantly to batch abends. Batch applications tend to be more computationally intensive than online applications. In practice, coding errors for initializations and computations are difficult to debug. Effects of initialization and computation enhancements will be felt later than problems that may result from other types of enhancements. Managers can use these results to watch for increases in abends resulting from software enhancements for initialization and computation. Not surprisingly, increased software complexity leads to increases in processing abends. Use of the two-stage model allows

examination of factors leading to increases in software complexity. Maintenance team characteristics and the use of purchased software can be influenced by managerial decisions. This, in turn, provides managers a means by which to reduce abends.

Although preliminary, the results of this research study make two contributions to work on software lifecycle maintenance. First, it is one of only a few longitudinal studies of software lifecycle maintenance processes, and an understanding of the longitudinal behavior of systems is increasingly important given the value and extent of legacy systems in use across the industry. The longitudinal data show the detailed patterns of types of maintenance activity beyond the three traditional high level categories. Second, this research offers a preliminary analysis of the link between lifecycle maintenance profiles and system performance. We have identified some variables contributing to the frequency of errors. By identifying some of the causes of abends, we can help managers better anticipate maintenance needs and recognize the consequences of prior enhancement activities.

## 5. REFERENCES


Banker, R. D.; Datar, S. M.; Kemerer, C. F.; and Zweig, D. "Software Complexity and Maintenance Costs," *Software Project Management: Readings and Cases*, C. F. Kemerer (ed.), Chicago: McGraw-Hill/R. D. Irwin, 1997, pp. 521-538.

Belady, R. A., and Lehman, M. M. "A Model of Large Program Development," *IBM Systems Journal* (15:1), 1976, pp. 225-252.

Dekleva, S., and Zvegintzov, N. "Real Maintenance Statistics," *Software Maintenance News* (9:2), 1991, pp. 6-9.

Gefen, D., and Schneberger, S. L. "The Non-Homogeneous Maintenance Periods: A Case Study of Software Modifications," *Proceedings of the IEEE Conference on Software Maintenance*, Monterey, CA, 1996.

Gill, G. K., and Kemerer, C. F. "Cyclomatic Complexity Density and Software Maintenance Productivity," *IEEE Transactions on Software Engineering* (17:12), 1991, pp. 1284-1288.

Greene, W. H. *Econometric Analysis*, 2nd ed., Englewood Cliffs, NJ: Prentice Hall, 1990.

*IEEE Standard for Software Maintenance*, New York: IEEE, 1993.

Kemerer, C. F. "Software Complexity and Software Maintenance: A Survey of Empirical Research," *Annals of Software Engineering* (1), September 1995, pp. 1-22.

Kemerer, C. F., and Slaughter, S. A. "An Empirical Approach to Studying Software Evolution," *IEEE Transactions on Software Engineering* (25:4), 1999, pp.1-17.

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

Nosek, J., and Palvia, P. "Software Maintenance Management: Changes in the Last Decade," *Journal of Software Maintenance* (2:3), 1990, pp. 157-174.

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

Robson, D. J.; Bennett, K. H.; Cornelius, B. J.; and Munro, M. "Approaches to Program Comprehension," *The Journal of Systems and Software* (14:2), February 1991, pp. 79-84.

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

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

Swanson, E. B., and Beath, C. *Maintaining Information Systems in Organizatoins*, New York: John Wiley and Sons, 1989.

Zvegintzov, N. "Software Should Live Longer," *IEEE Software* (15:4), 1998, pp. 19-21.