An Empirical Approach to Studying Software Evolution

Chris F. Kemerer and Sandra Slaughter

Abstract—With the approach of the new millennium, a primary focus in software engineering involves issues relating to upgrading, migrating, and evolving existing software systems. In this environment, the role of careful empirical studies as the basis for improving software maintenance processes, methods, and tools is highlighted. One of the most important processes that merits empirical evaluation is software evolution. Software evolution refers to the dynamic behavior of software systems as they are maintained and enhanced over their lifetimes. Software evolution is particularly important as systems in organizations become longer-lived. However, evolution is challenging to study due to the longitudinal nature of the phenomenon in addition to the usual difficulties in collecting empirical data. In this paper, we describe a set of methods and techniques that we have developed and adapted to empirically study software evolution. Our longitudinal empirical study involves collecting, coding, and analyzing more than 25,000 change events to 23 commercial software systems over a 20-year period. Using data from two of the systems, we illustrate the efficacy of flexible phase mapping and gamma sequence analytic methods originally developed in social psychology to examine group problem solving processes. We have adapted these techniques in the context of our study to identify and understand the phases through which a software system travels as it evolves over time. We contrast this approach with time series analysis, the more traditional way of studying longitudinal data. Our work demonstrates the advantages of applying methods and techniques from other domains to software engineering and illustrates how, despite difficulties, software evolution can be empirically studied.

Index Terms—Empirical methods, software evolution, software maintenance, longitudinal studies, time series, sequence analysis, gamma analysis, phasic analysis.

1 Introduction

If anything good can be said to have come out of the Year 2000 systems problem, it can be said 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 a consequence of this, there is greater understanding of how much of the cost of systems is due to their maintenance over time.

Unfortunately, while systems professionals have often gained this understanding, maintenance has continued to linger as a low status activity professionally, and an understudied phenomenon in the research community. A survey of empirical research in software maintenance reported that only 2 percent of empirical studies in software engineering focused on maintenance, despite reports that at least 50 percent of software effort is devoted to this activity [32], [39], [28]. Empirical software research presents the researcher with a number of obstacles, including the difficulties in collecting data and the lack of much existing theory and models. For maintenance, in addition to these 'normal' set of difficulties the researcher is confronted with additional concerns, for example, determining which parts of the system under maintenance have changed, as opposed to studying new development where, by definition, every artifact was created during the current project [30].

Within the world of software maintenance research empirical work on *software evolution* is particularly scarce. Consider two standard definitions for these terms:

- *Software Maintenance*: "...the *correction of errors,* and the implementation of *modifications* needed to allow an existing system *to perform new tasks*, and *to perform old ones under new conditions*..." [20] (emphasis added).
- *Software Evolution*: "...the *dynamic behavior* of programming systems as they are *maintained* and *enhanced* over their *life times*." [9] (emphasis added).

While maintenance refers to activities that take place at any time after the new development project is implemented, software evolution is defined as examining the dynamic behavior of systems, how they change over time. Given this definition, it is not surprising that empirical research on software evolution is scarce. The researcher has to collect data at a minimum of two different points in time. This creates practical difficulties in terms of sustaining support for the project over this period and/or finding an organization that collects and retains either relevant software measurement data or the software artifacts themselves [29].

The term software evolution is used in different ways by different researchers. While some use the term broadly to encompass both the initial development of the system and its subsequent maintenance, here we are focused exclusively on the events after initial implementation, consistent with its original focus [9]. Different researchers have

- *C.F. Kemerer is with the University of Pittsburgh, Pittsburgh, PA 15260. E-mail: ckemerer@katz.business.pitt.edu.*
- *S. Slaughter is with Carnegie Mellon University, Pittsburgh, PA 15213. E-mail: sandras@andrew.cmu.edu.*

Manuscript received 10 July 1998; revised 13 Feb. 1999.
Recommended for acceptance by T. Ross Jeffery.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 109540.

0098-5589/99/$10.00 © 1999 IEEE

segmented evolution in different ways. Perry focuses on three sources, or dimensions of evolution, domains, experience, and process [35]. Domains encompass the real world, the model of the real world reflected by the system, and any theories that support the model. All of these may change over time, forcing the system to similarly evolve. Experience is gained through use of the system over time and again may suggest reasons for the system to evolve. Finally, process, consisting of methods, technologies, and organizations also impact systems in use. An alternative, but in some ways similar breakdown is provided by Bennett [10]. He cites three items: alignment with organizational objectives, process issues, and technical issues as categories of focus within software evolution.

Both authors cite the work of Belady and Lehman as seminal to their arguments. A key notion behind that work is the concept of software system *entropy*. The term entropy, with a formal definition in physics relating to the amount of energy in a closed thermodynamic system that is unavailable for work, has itself evolved over time to be used broadly as a measure of any system's disorder or uncertainty.[1] As such, this term has had significant appeal to software maintenance researchers since it suggests a set of reasons for software maintenance, given that a casual observer might not otherwise expect software to require maintenance in the common usage of the term to imply repair, e.g., automobile maintenance, or even computer hardware maintenance.

In this paper, we focus on the research tasks and methods involved in our large scale, multiyear empirical study of software evolution. While structured, to the degree possible, as a typical research paper, with an introduction, prior research, problem statement, methodology, results, and discussion section, this paper differs from a typical paper in that the focus is on the research methods rather than the results or even the research questions or models. Given both the practical importance of, and the scarcity of empirical research in, software evolution we feel that documenting our research process may be particularly valuable to the empirical software engineering community.

1.1 Empirical Research in Software Evolution

While we have had considerable experience in conducting software maintenance research as defined above, this program of research represented our initial work in empirical study of software evolution. As such, the first step was to review the literature in the software evolution area. As described above, a number of conceptual and practical obstacles conspire to make empirical software evolution research difficult. Therefore, there is a relative paucity of empirical work in the area. However, we found a number of prior pieces of work, all of which are summarized below, with particular attention paid to the research method issues, consistent with the goal of this special issue.

1. Note that entropy has been adopted in other contexts, particularly information theory, which will not be applied here. However, for an example of this work in a software context, see [24], [14]. Entropy halso been claimed as the reason for lack of order in object oriented class hierarchies ("conceptual entropy") [20].

1.1.1 Belady and Lehman

The seminal empirical work in this area is by Belady and Lehman [9]. Their work in the '70s involved studying 20 releases of the OS/360 operating systems software, perhaps the first empirical research to focus on the dynamic behavior of a relatively large and mature (12 years old) system. They made a number of observations about the size and complexity growth of the system, which led them to postulate five "laws" of software evolution dynamics: Continuing Change; Increasing Complexity; The Fundamental Law of Program Evolution; Conservation of Organizational Stability, and Conservation of Familiarity. These laws were further developed in a paper published in 1980 [32]. In this single-authored work Lehman elaborates on the meaning of the laws in the context of real systems. Empirical data are presented relating to a small number of later releases of a general-purpose batch operating system.[2] First, it is argued, through graphical presentation of the data, that the five laws of evolution are supported. Then, a detailed analysis is given to illustrate how the knowledge gained through the analysis of the first 19 releases of the software can be used to plan release 20. It demonstrates how some initial estimates were likely to be optimistic or otherwise inaccurate, until corrected with access to the historical data, and is a useful demonstration of the practical application of this type of research to software maintenance management.

1.1.2 Yuen

The first systematic study of Belady and Lehman's laws was done by one of Lehman's students, Chong Hok Yuen. His work was presented in a series of three empirical papers, published in 1985, 1987, and 1988 [15], [16], [17]. In [15], 19 months of "bug" (defect) data from a large operating system were analyzed. Data were analyzed for four different months (at 1, 7, 13, and 19 months). Seven dependent variables are described, but only two sets of results, priority class (severity) and response time, are described. The author reports a number of results relating to defect discovery and correction, including the observations that defects tend to be discovered in batches, and tend to be clustered shortly after each release. Interestingly, the time taken to fix a defect did not increase with time, as might be expected due to complexity growth of the system. The author suggests that his research can be used to detect improving/deteriorating situations by observing a trend in the response time or in the number of outstanding fixes and thereby allow managers to take actions to improve process and system performance. In [16] Yuen tests Belady and Lehman's five laws of evolution dynamics. He re-examines three different systems from Belady and Lehman, plus several other systems, and looks at a variety of dependent variables, including the number and percentage of modules handled. The data are evaluated using the Runs, Turning Points, and Phase Length tests. After reexamining the data from previous studies, the characteristics observed for OS/360 did not necessarily hold for other systems; in particular, while the first two laws were supported, the remainder of

2. It appears that some of these data are also analyzed by Yuen in his work. See citations below.

the laws were not. However, the author notes that these latter laws are more based upon those of the human organizations involved in the maintenance process rather than the properties of the software itself. In [17] Yuen continues his examination of data from the "B" operating system, this time conducting time series analysis on the number of "notices." These notices were information issued to the commercial users of the system, detailing how to avoid, circumvent or patch the error which it describes until the error is properly mended in a subsequent official release of the operating system. The author notes that previous studies of what he terms "global" maintenance data observed at the "global level" tend to show few patterns, if any. These results were previously interpreted as "invariance" and "well-defined averages" and led to Evolution Dynamics. This paper examines the maintenance of a large piece of software at a "sublevel," as well as "global level," to understand how maintenance observed at the "sublevel" contribute to the observations made at the "global level." Data were analyzed for 168 calendar week data points, with five releases. The dependent variable is the number of "notices" per week. In addition to the tests from [16], the author also uses Time Series Analysis/Spectral Analysis techniques (correlogram, chi-square, autoregression, autoregression moving average) as well as linear filtering (moving average technique). The author reports a variety of results, including the fact that external factors seem to play a large role in determining the amplitude of each peak and the interval between successive peaks.

1.1.3 Tamai and Torimitsu

Tetsuo Tamai and Yohsuke Torimitsu used a questionnaire survey of Japanese organizations to examine business application software replacements over a 5-year period. They report a number of descriptive results for their 95 data points, including the fact that smaller scale software tends to have a shorter life, and that administrative type applications (e.g., personnel, accounting) tend to have longer life than business supporting type applications (e.g., sales support, manufacturing).

1.1.4 Cook and Roesch

Cook and Roesch examined 10 releases over 18 months of a real time telephone switch from a German telecommunications firm [19]. Their primary focus was on an exploratory factor analysis of software metrics for complexity, and in particular, the authors conclude that information metrics performed relatively the best vs. other metrics, such as Halstead and McCabe metrics, and simple lines of code. However, they also examined the evolution of the system over 18 months of data, and argued that they found support for the laws of evolution, citing continuing change, increasing entropy, and total change not being uniform. The system grew from 223 functions to 359 functions during the period studied.

1.1.5 Gefen and Schneberger

Gefen and Schneberger explored two distribution patterns of software maintenance modifications (constant and decreasing) to determine if the software maintenance distribution is homogeneous [22]. The authors studied Software Problem Reports (SPRs) from a 4GL system. Monthly data (29) points were collected. These SPRs were characterized by modification type (corrective or adaptive), plus the number of new "applications." In addition, they tracked the number of modifications caused by previous modifications. They noted that the rate of maintenance modifications decreases over time in the aggregate, but not if viewed in individual phases, which they describe as "stabilization," "improvement," and "expansion."

1.1.6 Basili et al.

A relatively more recent study by Basili et al. examined 25 software releases of 10 different systems at NASA Goddard, including over 100 software systems totaling about 4.5 million LOC [8]. A focus of the study was to characterize the types of maintenance activities and examine both the total effort and the effort distributions across these maintenance projects. Data collection lasted 18 months. The study looked at the three Swanson maintenance change types (corrective, adaptive, perfective) and a set of maintenance activities for each. They found that error correction efforts, typically small changes, required significant isolation activity, while enhancements required more time on inspection and certification. Effort for design and coding and unit testing (CUT) were similar for the two types. Statistical analysis of these differences was necessarily limited. Pie charts on effort percentages by task type by system and on Swanson typology are presented. Similar to earlier work in new development, this paper argues against small releases, presumably due to scale economies [6], [5], [7].

1.1.7 Lehman et al.

Most recently, Lehman and his colleagues have begun a new series of investigations into software evolution, labeled the FEAST project [33]. In an early paper, they describe an empirical analysis of an 8-year old financial transaction system. Of the total of 100 releases (including very small ad hoc releases for specific clients), the researchers examined 21 releases spanning 5 years of evolution. For each release they noted its size in terms of number of modules and the number of modules changed. In addition, some statistical modeling was also done, developing a growth model for module size per release using an inverse square model. One of the results of this model was that the observed pattern of system growth was established by the sixth release. Interestingly, as in the case of the earlier OS/360 study, the final few observations are seen as outliers to the generally smooth growth model. This paper also updates and summarizes what are now eight laws of software evolution.

All of these previous studies are summarized in Table 1. Briefly, the overall area can be summarized in a number of ways. First, and foremost, is the general paucity of any kind of empirical work. As noted above, this can be attributed to a variety of practical concerns evidenced in all empirical work in software maintenance, plus the additional obstacles facing software evolution work. And, within that small set of work, the greatest contribution has been made by Belady, Lehman, and their student, Yuen, suggesting that the total

TABLE 1
Empirical Software Evolution Research

Author	Publication	Methodology	Data	Dependent Variables	Statistical Test
Belady & Lehman (1976)	IBM Systems Journal	Field study	21 user-oriented releases	Release sequence numbers, system age, system size, number of system modules, complexity	Multivariate regression, Auto-correlation
Yuen (1985)	IEEE Conference on Software Maintenance	Field study	5,000 "components" over 19 months period, 3,000 KLOC	Priority class, originator's reference, release affected, component affected, machine affected, category of error discovered, response time	Chi-square, Contingency coefficient measure, Time series, T-statistic, Auto and cross-correlations, Poisson distribution
Yuen (1987)	IEEE Conference on Software Maintenance	Secondary data analysis	Modules from OS/360, OMEGA, EXECUTIVE, BD, B, DOS, CCSS systems	Cumulative modules handled, handle rate, fraction of modules handled, size, release interval, net growth	Runs test, Turning points test, Phase length test
Yuen (1988)	IEEE Conference on Software Maintenance	Field study	"notices" - information issued to the commercial users of the system using same data set as Yuen (1985)	Releases and number of "notices" per week	Runs test, Turning points test, Phase length test, Time series analysis/Spectral analysis techniques, Linear filtering
Tamai & Torimitsu (1992)	IEEE Conference on Software Maintenance	Survey	95 systems from various organizations reporting on work done in prior 5 years. Mainframe software, 70% COBOL.	Age of software life span, software size before and after replacement, application areas, replacement factors	Sample statistic, Correlation
Cook and Roesch (1994)	Journal of Systems and Software	Field study	10 versions of real time German telephone switching software released over 18 months.	Number of functions, number of functions changed, number of major changes	Correlations, exploratory factor analysis with varimax rotation
Gefen & Schneberger (1996)	IEEE Conference on Software Maintenance	Field study	29 months of Software Problem Reports (SPRs), 250 KLOC	Modification type (total number of SPRs, number of corrective SPRs, number of adaptive SPRs), number of new applications, number of modifications caused by previous modifications	Linear regression, Wilcoxon Matched-Pairs Signed-Ranks Test, Kolmogorov-Smirnov Goodness of Fit Test
Basili, et al. (1996)	IEEE International Conference on Software Engineering (ICSE)	Field study	25 software releases of 10 different systems at NASA Goddard.	Effort and size for different types of maintenance activities/tasks.	Mann-Whitney U non-parametric test; OLS regression
Lehman, et al. (1997)	International Software Metrics Symposium	Field study	21 software releases of a financial package	Size of system in modules and number of modules changed	Least squares and inverse square regression model; mean absolute error

amount of completely independent research is even smaller than Table 1 would suggest.

Further examination of Table 1 reveals that prior researchers have generally been limited in the sophistication of their analysis due to the small sample sizes created by focusing on, for example, system releases as the unit of analysis. This means that typically only single variable OLS regression models have been estimated. Exceptions to this, including Yuen, employ time series analysis on larger data sets. However, generally few results were generated by his time series analyses. (Some possible reasons for this are suggested in the results section of the current paper.) Finally, the work has been disseminated almost exclusively through conference proceedings, and within those, primarily the Conference on Software Maintenance, which are generally less accessible to researchers interested in software evolution than archival journals.

2 Research Approach

2.1 Overview

Having completed the above review of the literature, we designed our research program to maximize its contribution to this stream of work. First and foremost, we were interested in conducting a longitudinal empirical study of software evolution, as much of the prior work in this area has been cross-sectional, limited in scope, or conceptual. Second we preferred to study actual business systems in real organizations. It is valuable to observe how software evolves in the real world because this observation can be used to help build theories of software evolution that can be generalized to other commercial organizations. Finally, we sought a research site that collected rich, detailed, high quality change history data on software systems over time. From the point of view of research methods, we were most interested in those that exploited the longitudinal and detailed nature of the data we were planning on collecting.

2.2 Site Selection and Data Evaluation

We selected a large United States midwestern retailer (hereafter referred to as the Retailer) as our research site. This site is ideal in terms of meeting our objectives. The Retailer has a large, centralized Information Systems (IS) department that handles information processing for all of its various department stores. The centralized location provides a convenient point of access for data collection. The Retailer's IS department has separate development and maintenance units. This organizational design contributes to the Retailer's ability to better control, manage, and measure maintenance activities [44]. The maintenance unit tracks longitudinal data relating to changes, costs, and effort to maintain all of the systems under its area of

```
*-------------------------------------------------------------*
 PROGRAM-ID. <REDACTED>M110.
 AUTHOR. JOHN <REDACTED>.
 INSTALLATION. <REDACTED>.
 DATE-WRITTEN. FEBRUARY 1990.
 DATE-COMPILED.
*
*****************************************************************
*  XXXM110      ON-LINE RECEIVING ENTRY PROGRAM         CHANGE LOG
*****************************************************************
* DATE: 01/02/91
* PROGRAMMER: JOHN <REDACTED>
* CHANGE: COMPLETELY RESTRUCTURED PROGRAM.
* CHANGE: RESET IDOC MRNC-DEDUCT-FLAG, WHENEVER A RECEIPT IS MADE,
*         TO A 'N'O VALUE.
* CHANGE: REDUCE THE NUMBER OF SKU LINES ON THE SCREEN TO EIGHT
*         AND INSTEAD HAVE IDOC COMMENT FIELDS.
* CHANGE: BE SURE NEXT PO IS NEVER SET TO AN SAV PO.
* CHANGE: UPDATE BTCN SEGMENT FOR EVERY UPDATE TRANSACTION, EVEN
*         IF USER DOES NOT ENTER END-OF-RCPT = 'Y'.
* CHANGE: ADDED FEATURE THAT LOSS-DAMAGE NUMBER AND DEBIT-MEMO
*         NUMBERS ON BTCN SEGMENT WILL NOT BE REPLACED WITH ZEROS
*         FROM A CURRENT UPDATE IF THEY HAD CONTAINED NON-ZEROS.
* REQUEST #: 403
****************************************************************
* DATE: 02/26/91
* PROGRAMMER: JOHN <REDACTED>
* CHANGE: FIX LOOP BUG.
* CHANGE: DON'T INSERT BMRR SEGMENTS FOR MANIFESTS.
* REQUEST #: EMERGENCY
****************************************************************
* DATE: 02/27/91
* PROGRAMMER: JOHN <REDACTED>
* CHANGE: BTCN RECEIPT DATE WAS NOT BEING PROPERLY SET.
* REQUEST #: EMERGENCY
****************************************************************
* DATE: 04/05/91
* PROGRAMMER: D.A.<REDACTED>
* CHANGE: RESET IDOC-TYPE FLAG (INDICATING ACTIVE PO) AND
*         IDOC-RECEIPT-FLAG WHEN UPDATING IDOC-MRNC-DEDUCT-FLAG.
* REQUEST #: PROBLEM SHEET.
****************************************************************
```

Fig. 1. Portion of a sample change log.

responsibility. Many maintenance personnel have worked only in the unit for a long time (on average, 8-10 years), and many of the maintenance supervisors have been with the unit since it started in the '70s. This stability of personnel and focus on tracking maintenance performance results in databases with high quality data on software evolution that extend over a period of more than 20 years.

2.3 Data Collection

We extracted software change events from histories or logs that were written by maintenance programmers each time they updated a software module in the application portfolio. Logs were kept for more than 25,000 changes to 3,800 software modules in 23 different COBOL business systems from the beginning of the early '70s, when many of the systems were originally written, until the present. The COBOL systems represent more than two-thirds of the functionality accomplished by the Retailer's information systems portfolio. The kind of data available in the change logs includes the original software module creation data and author, the function of the module, the programmer making a change, the date of the change, and the description of the change. For an example of such a change log, see Fig. 1.

Such documentation allows the unit of analysis for this research to be the change event, of which there were approximately 25,000 during this period. Note that this creates the opportunity for a much larger sample size than has been the case in empirical studies of this type, which have typically studied on the order of 25 units [8], [9], [22], [33].

2.4 Data Transformation

Change logs were used to identify and classify change events for each software module in a system. We developed a coding scheme to categorize each change event that relies upon and extends the standard industry categorization for software maintenance activities [26]. Maintenance events were initially categorized into three basic types: corrections, adaptations, and enhancements [43]. We further refined and elaborated these three basic maintenance types into 30 subcategories (Table 2) that were drawn from classification

TABLE 2
Classification Scheme (Codes in Parentheses)

Corrective	Enhancement/Perfective
Data Handling *(CorrectDat)*	Data Handling: Add, Change, Delete
Logic/Structure *(CorrectLog)*	*(EnhDatAdd, EnhDatChg, EnhDatDel)*
Computation *(CorrectCom)*	Logic/Structure: Add, Change, Delete
Initialization *(CorrectInit)*	*(EnhLogAdd, EnhLogChg, EnhLogDel)*
User Interface *(CorrectUserI)*	Computation: Add, Change, Delete
Module Interface *(CorrectModI)*	*(EnhComAdd, EnhComChg, EnhComDel)*
Adaptive	Initialization: Add, Change, Delete
Data Handling *(AdaptData)*	*(EnhIniAdd, EnhIniChg, EnhIniDel)*
Logic/Structure *(AdaptLogic)*	User Interface: Add, Change, Delete
Computation *(AdaptComp)*	*(EnhUsrIAdd, EnhUsrIChg, EnhUsrIDel)*
Initialization *(AdaptInit)*	Module Interface: Add, Change, Delete
User Interface *(AdaptUserI)*	*(EnhModIAdd, EnhModIChg, EnhModIDel)*
Module Interface *(AdaptModI)*	**New Program** *(NewProgram)*

schemes from prior research for analyzing maintenance activities [12], [37].

To classify each event in a change log, we adopted a content analytic approach (Krippendorff, 1980) using a combination of latent and manifest coding techniques. Manifest coding involves looking through the text of the change log for visual occurrences of certain keywords. Latent coding identifies the underlying meaning in text of the change log when keywords are not sufficient to categorize events. Both approaches to coding were necessary, as the maintenance programmers were not always completely consistent in describing maintenance events when they logged their maintenance activities.

2.5 Data Validity and Reliability

We employed three independent coders[3] to content analyze the change logs. The coders were chosen for their in-depth knowledge of the information systems field so that they could identify terms and acronyms and categorize events accurately. Because of the sensitivity of data dependent research to error, it is important that measures be as reliable and valid as possible.

We, therefore, implemented a number of techniques and procedures to maximize inter-rater reliability and to assess and improve coding validity. A coding flowchart (Fig. 2) was developed and helped to provide a consistent way to classify change events. A standard set of coding procedures and training materials was developed, and each coder was instructed in the procedures. Finally, coders were instructed to refer any change event that could not be classified using the flowchart to the researchers for resolution.

3. Note that in this context "coder" refers to a research assistant who codifies data, not to a computer programmer, to whom the term coder is often applied and whose usage may be more familiar to *IEEE-TSE* readers. However, all three coders also had programming experience.

To help ensure consistency between the coders, several trial data coding processes were performed. In these trials, each coder independently coded the same maintenance events that were randomly selected by the researchers from the change logs. After the independent coding, the Cohen coefficient of agreement or Cohen's K for nominal scales was computed to assess the relative pairwise agreement between the coders [18]. Systematic differences in coding after each trial were discussed and resolved, and the coders independently coded another set of maintenance events. After the first trial round of independent coding, the intercoder reliability (Cohen's K) averaged 0.42, indicating moderate agreement [31]. After the second trial round of independent coding, the average Cohen's K improved to 0.61. After the third trial round of independent coding, the average value for Cohen's K exceeded 0.72, indicating substantial agreement. Given the complexity and length of our coding scheme, the reliability and agreement between coders are both quite high and are consistent with numbers reported in other empirical studies of this nature [27], [41], [42], [40]. In addition, there was no correlation between individual coders and the proportions of events assigned to different codes, suggesting that the coders interpreted the events in an unbiased manner. Subsequently, the maintenance events for the different software systems were divided equally between the coders, and the events were coded independently. To test for decrements in reliability due to "coder drift" or exhaustion, we checked a sample of each coder's work near the end of the coding process against the second author's coding of the same events. We found no evidence of a decrease in reliability over time. And as a final validity check, the researchers randomly inspected coded events, and did not find degradations in accuracy. Fig. 3 illustrates a coded change log.

After coding the maintenance events for the modules in a system, the coders entered the data into spreadsheets that

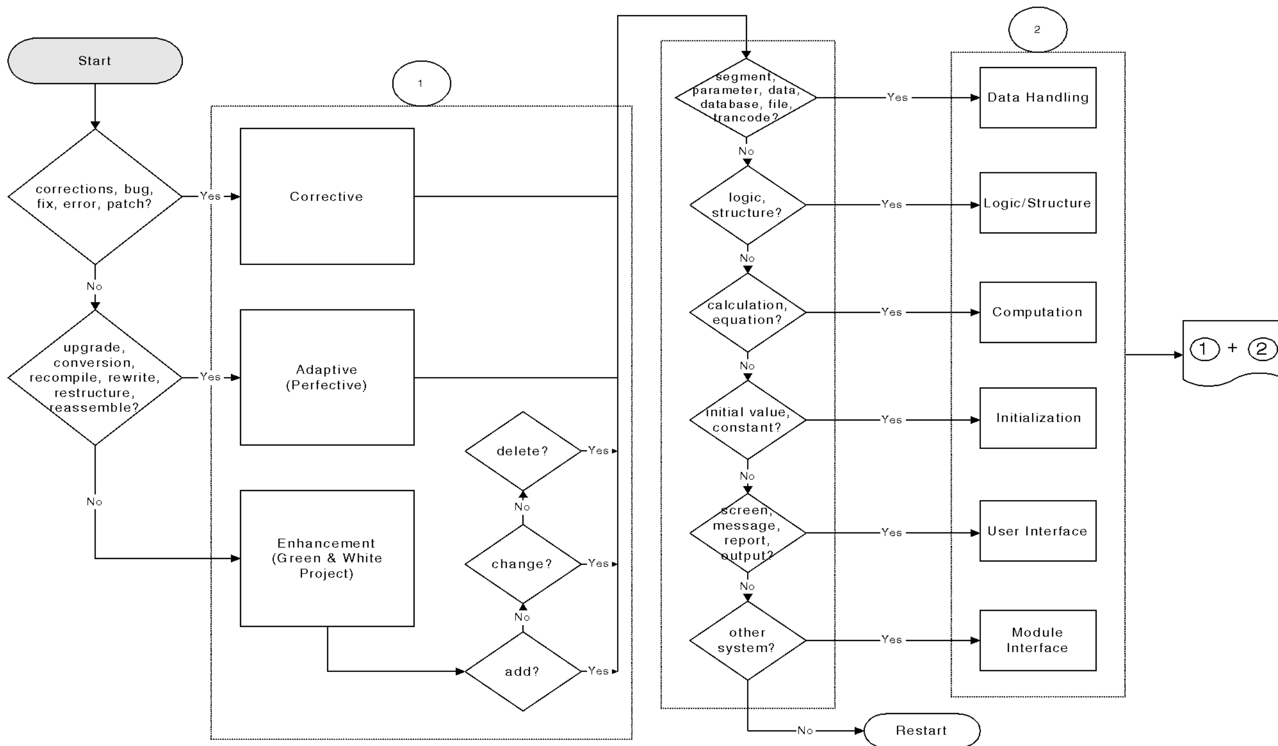

Fig. 2. Coding flowchart.

were later imported into relational databases. These relational databases were designed and implemented to hold the contents of the coded maintenance events. A database approach was selected as an efficient and flexible way to store the vast amounts of change event data resulting from the coding, and enabled the running of queries and searches. In addition, the flexibility of an entity relationship design facilitated creating different views of the data as well as exporting data in formats suitable for a variety of software packages for further analysis.

2.6 Descriptive Data—Two Representative Systems

While the focus of this article is on the research approach, we present here some descriptive results relating to the change events for two representative systems, the Financial Sales Reporting System and the Manifest Shipping System. A descriptive profile of the systems is presented in Table 3.

The distribution of change events for these two systems is presented in Table 4.

Overall, over 81 percent of the changes to the Financial Sales Reporting System were either enhancements or new programs, and the same figure was over 83 percent for the Manifest Shipping System. This is consistent with much prior research, and supports the need for determining more detailed change event categories, since the traditional three-way classification is relatively coarse. In particular, our 30-category classification scheme appears to be appropriate for these systems, as each system had examples of 24 of the 30 possible categories represented.

But, beyond these broad notions, a number of more specific observations can be made about the descriptive data in Table 3. The top two detailed categories in terms of frequency for both systems were enhancements to logic and to data handling. In sum, these two categories account for 38 percent and 44 percent of the total change events for each system. The top 10 categories in terms of frequency are the same for both systems, but in somewhat different sequence. Some differences include:

- enhancements to the user interface occur twice as often (percentage-wise) for the Manifest system as for the Financial system
- corrections and adaptations to data handling occur twice as often for the Financial system as for the Manifest system
- new programs occur twice as often for the Financial system as for the Manifest system
- corrections and enhancements to computations occur more than twice as often for the Financial system as for the Manifest system.

In addition to the change events, other detailed data were captured about the systems (presented in Table 3). A number of observations can be made about the two systems from these data.

- The Financial system is twice as old and has more than twice as many modules as the Manifest system. However, most of the Financial system's modules are batch, while more than half of the Manifest's modules are on-line (interactive) programs.
- In terms of lines of code, the Manifest system is larger, even though it has fewer than half as many modules. As a result, the average module size for the Manifest system (3,878 lines of code) is more than twice that for the Financial system (1,666 lines of

```
*--------------------------------------------------------------*    CODE
 PROGRAM-ID. XXXM110.
 AUTHOR. JOHN <REDACTED>.
 INSTALLATION. <REDACTED>.
 DATE-WRITTEN. FEBRUARY 1990.                                       NewProgram
 DATE-COMPILED.
*
*****************************************************************
*  XXXM110      ON-LINE RECEIVING ENTRY PROGRAM        CHANGE LOG
*****************************************************************
* DATE: 01/02/91
* PROGRAMMER: JOHN <REDACTED>
* CHANGE: COMPLETELY RESTRUCTURED PROGRAM.                          AdaptLogic
* CHANGE: RESET IDOC MRNC-DEDUCT-FLAG, WHENEVER A RECEIPT IS MADE,  EnhDatChg
*         TO A 'N'O VALUE.
* CHANGE: REDUCE THE NUMBER OF SKU LINES ON THE SCREEN TO EIGHT     EnhUsrIChg
*         AND INSTEAD HAVE IDOC COMMENT FIELDS.
* CHANGE: BE SURE NEXT PO IS NEVER SET TO AN SAV PO.                EnhLogChg
* CHANGE: UPDATE BTCN SEGMENT FOR EVERY UPDATE TRANSACTION, EVEN    EnhDatChg
*         IF USER DOES NOT ENTER END-OF-RCPT = 'Y'.
* CHANGE: ADDED FEATURE THAT LOSS-DAMAGE NUMBER AND DEBIT-MEMO      EnhLogAdd
*         NUMBERS ON BTCN SEGMENT WILL NOT BE REPLACED WITH ZEROS
*         FROM A CURRENT UPDATE IF THEY HAD CONTAINED NON-ZEROS.
* REQUEST #: 403
****************************************************************
* DATE: 02/26/91
* PROGRAMMER: JOHN <REDACTED>
* CHANGE: FIX LOOP BUG.                                             CorrectLog
* CHANGE: DON'T INSERT BMRR SEGMENTS FOR MANIFESTS.                 CorrectDat
* REQUEST #: EMERGENCY
****************************************************************
* DATE: 02/27/91
* PROGRAMMER: JOHN <REDACTED>
* CHANGE: BTCN RECEIPT DATE WAS NOT BEING PROPERLY SET.             CorrectDat
* REQUEST #: EMERGENCY
****************************************************************
* DATE: 04/05/91
* PROGRAMMER: D.A.<REDACTED>
* CHANGE: RESET IDOC-TYPE FLAG (INDICATING ACTIVE PO) AND           CorrectDat
*         IDOC-RECEIPT-FLAG WHEN UPDATING IDOC-MRNC-DEDUCT-FLAG.
* REQUEST #: PROBLEM SHEET.
****************************************************************
```

Fig. 3. Examples of a coded change log.

code). In terms of normalized complexity metrics (McCabe's and Halstead's primitives), the Financial system is more complex per line of code. That is, the Financial system has higher decision complexity, more unique and total operators, and more unique and total operands per line of code when compared to the Manifest system. This implies that the Financial system is more data and computationally complex (intensive).

- The Financial system costs 9-10 times more on average per change, per FP and per LOC to modify (enhancements, corrections, and adaptations) than does the Manifest system. The Financial system costs 5-7 times more per enhancement, per FP and per LOC to enhance than does the Manifest system. The differences are quite extreme for maintenance (corrections and adaptations); the Financial system costs 16-28 times more to maintain (i.e., correct and adapt) per maintenance, per FP and per LOC than does the Manifest system.

2.7 Data Analysis—Conventional Approach (Time Series)

We first considered time series analysis for analysis of our change event histories. As indicated earlier, time series analysis has been employed in past studies by Yuen of software evolution [15], [16], [17]. In time series analysis, continuous data are required, and techniques such as ARIMA (autoregressive integrated moving average) are used to fit models which indicate systematic patterns in the data by Box et al. [11]. We analyzed the number of changes to software systems by time period (which we identified as a month) to see whether we could model and predict patterns in the number of changes over time in a system. We found that selection of a year as the basis for the time period was too coarse a distinction, and did not result in a sufficient number of time periods for analysis. But, on the other hand, selection of day or week as the basis for the time period was too fine a distinction, and did not result in sufficient occurrence of changes for analysis. Fig. 4

TABLE 3
Descriptive Profile of Two Systems

		Financial Sales Reporting System	Manifest shipping System
Age Size	Age (years)	20	10
	Size in Total Lines of Code (LOC)	181,652	189,999
	Size in Function Points (FP)	1,321	1,446
	Number of Modules	109	45
	% of Online Modules	7%	62%
	% of Batch Modules	93%	38%
	Average Module Size (LOC per module)	1,666	3,878
Complexity	Cyclomatic Complexity per LOC	0.04624	0.04296
	Unique Operators per LOC	0.02558	0.01006
	Unique Operands per LOC	0.21336	0.13748
	Total Operators per LOC	0.74307	0.59015
	Total Operands per LOC	0.73631	0.69441
	Calls per LOC	0.00459	0.01474
Average Cost	Average Cost per Change	$2,109.01	$208.15
	Average Cost per FP	$96.88	$11.13
	Average Cost per LOC	$0.71	$0.08
	Enhancement Cost per Enhancement	$1,340.70	$177.72
	Enhancement Cost per FP	$40.25	$8.52
	Enhancement Cost per LOC	$0.29	$0.06
	Maintenance Cost per Maintenance	$5,828.63	$374.31
	Maintenance Cost per FP	$69.14	$2.43
	Maintenance Cost per LOC	$0.44	$0.02

illustrates an example of the cumulative number of changes plotted by month for one of the older systems, a Financial Sales Reporting System that was more than 20 years old. As can be seen in this graph, even with a monthly time unit, there are many time periods in this series where there are no changes occurring. This data present a problem in terms of estimating a time series model as the series is not *stationary*, i.e., does not have the same mean and variance throughout [11]. This time series could not be transformed to be sufficiently stationary despite attempting three standard transformations: differencing, logarithmic, and square root. Since the identification process for the autoregressive and moving average models requires stationary time series, we could not estimate these models using the data for this system.

Fig. 5 illustrates a different problem that we encountered using conventional time series approaches. This figure plots the number of monthly changes to a relatively younger, but more volatile system, the Manifest Shipping System. As can be inferred from this graph, the series does not appear to be stationary, but we were able to transform the data using first differencing into a stationary series. We determined from the autocorrelation and partial autocorrelation functions that a simple ARIMA (0, 1, 1) model was appropriate. The coefficient for this moving average model was estimated using maximum likelihood and was significant ($\theta = 0.03,\ \mathrm{t} = 21.52,\ \mathrm{p} = 0.00$). For this model, the value of θ implies that the difference between consecutive observations in the series equals the current random disturbance minus approximately *zero* times the previous disturbance. The closeness of the coefficient to zero implies that this process approximates a "random walk" because the difference between values is a random step away from the difference between previous values.

Therefore, we found that conventional time series models did not provide much insight into the software evolution process for our data, either because the data were not stationary or because the data series occurred in a largely random fashion.[4] Furthermore, and perhaps most seriously for our future research into software evolution, we found that time series approaches with the focus on relatively simple, continuous, quantitative variables did not exploit the richness of our coding categories for the data.

2.8 Data Analysis—Alternative Approach (Sequence Analysis)

An alternative to time series analysis is an approach from the social sciences called sequence analysis [2]. Sequence analysis consists of the identification and testing of a phasic model or explanation that accounts for the characteristics of a long series of time-ordered observations. It is based on analysis of categorical data, rather than continuous data. The objective of sequence analysis is to identify sequences of acts or phases in the developmental pattern of behavior. It results in a map of the interaction consisting of a series of

4. Note that Yuen reaches similar conclusions about the limitations of time series methods in his analysis [17].

TABLE 4
Distribution of Change Events for Two Systems

Financial Sales Reporting System				Manifest Shipping System		
Event	frequency count	% of total		event	frequency count	% of total
EnhLogChg	164	19.50%		EnhLogChg	152	23.31%
EnhDatChg	157	18.67%		EnhDatChg	138	21.17%
NewProgram	109	12.96%		EnhDatAdd	79	12.12%
EnhDatAdd	82	9.75%		CorrectLog	49	7.52%
CorrectLog	48	5.71%		NewProgram	45	6.90%
EnhLogAdd	43	5.11%		EnhLogAdd	44	6.75%
CorrectDat	41	4.88%		EnhUsrlChg	40	6.13%
AdaptData	32	3.80%		CorrectDat	17	2.61%
EnhDatDel	29	3.45%		EnhDatDel	14	2.15%
EnhUsrlChg	24	2.85%		EnhLogDel	13	1.99%
EnhLogDel	21	2.50%		AdaptData	13	1.99%
CorrectCom	21	2.50%		EnhUsrlAdd	8	1.23%
EnhUsrlAdd	19	2.26%		CorrectUsr	8	1.23%
EnhComChg	14	1.66%		AdaptLogic	8	1.23%
CorrectUsr	8	0.95%		EnhUsrlDel	4	0.61%
EnhComAdd	7	0.83%		CorrectInit	4	0.61%
EnhComDel	5	0.59%		AdaptUser	4	0.61%
AdaptUser	5	0.59%		EnhInitDel	3	0.46%
EnhModChg	4	0.48%		CorrectCom	3	0.46%
EnhModAdd	2	0.24%		EnhInitChg	2	0.31%
EnhInitAdd	2	0.24%		EnhModlAdd	1	0.15%
CorrectMod	2	0.24%		EnhInitAdd	1	0.15%
EnhInitDel	1	0.12%		AnhComChg	1	0.15%
CorrectInit	1	0.12%		CorrectMod	1	0.15%
Total	841	100.00%		Total	652	100.00%

coherent periods or functions following one on another. Sequence analysis can address questions concerning the types of development paths, the structural properties of a process, and the factors influencing different types of development paths. Sequence analysis has been used to study such phenomena as diverse as implementation events in information systems projects [38], group problem solving strategies [21], hostage negotiations [25], and the career paths of musicians [4].

In our study, we are interested in learning how systems evolve over time based upon a rich set of detailed change events. Specifically, we seek to identify patterns in the change events, to determine whether these patterns coalesce into phases or paths along which systems progress, to understand whether systems follow similar evolutionary paths, and to analyze why systems may follow different evolutionary paths. Our intent, therefore, is to study and develop a theory of the *process* of software evolution. Given this objective, sequence analysis which focuses on understanding a process is a more appropriate methodology than the more traditional variance-oriented approaches that focus on identifying cause-effect relationships such as regression analysis [3]. It is also possible to link process and variance methodologies, when the intent is to determine the causes and effects of different process typologies [1]. We focus here on sequence analysis methods as we wish to first understand the process and patterns of software evolution. The identification, ordering, and comparison of evolutionary paths can be accomplished using three sequence analytic methods: phase mapping, gamma analysis, and gamma mapping.

2.9 Phase Mapping

Phase mapping is a method that is applied to sequentially ordered events and that groups "like" events into discernible phases. The starting point for phase mapping is a series of time-ordered categorical event codes. In our study, the event codes correspond to labels for the change events in our coding scheme (Table 2), and are sorted by date of occurrence. Fig. 6 shows the first 20 change events for the Financial Sales Reporting System and for the Manifest Shipping System.

The ordered events are then "parsed" into discrete phases based upon the assumption that phases are indicated by consecutive occurrence of a number of events of the same type. We used a flexible phase mapping procedure to detect phases in our data. Under this procedure, a phase is defined when there are at least three

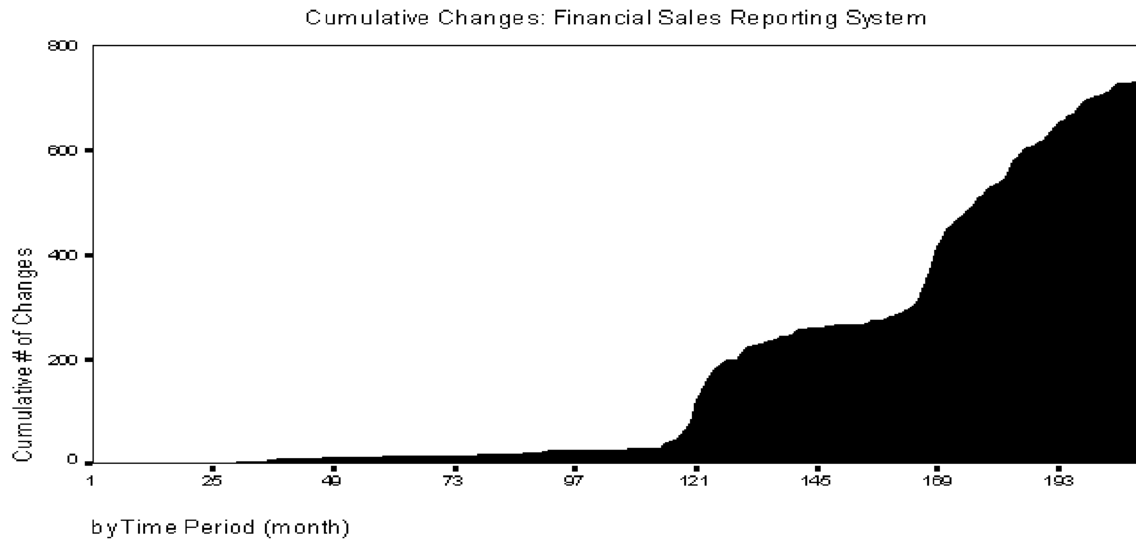

Fig. 4. Financial sales reporting system.

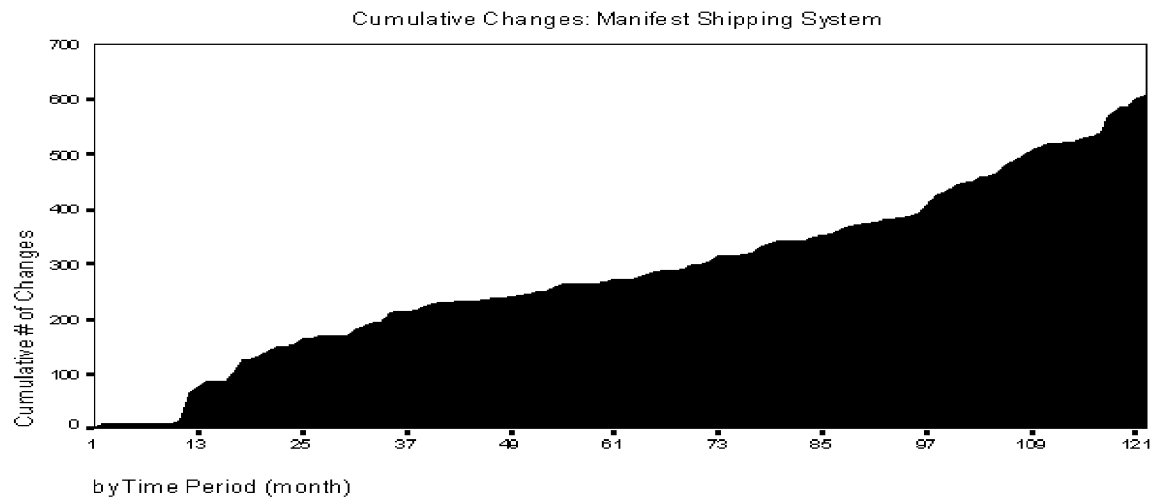

Fig. 5. Manifest shipping system.

consecutive events that share the same event type. The length and complexity of the event data influence the choice of the number of events that defines a phase. For our data, we discovered that a rule of minimally four consecutive occurrences of an event to identify a phase was most appropriate (a rule of minimally three consecutive occurrences resulted in too many detailed phases, and five resulted in too few phases) [34], [36].

The flexible phase mapping procedure produces a detailed phase map that describes phases which can be of different lengths, identifies phase recycling or repetition, and identifies "null" or disorganized periods that do not appear to cohere as phases. We used the *Winphaser* software[5] to conduct the mapping procedure, with our rule of minimally four consecutive event occurrences to identify a phase. We then manually refined the maps created by *Winphaser* to label the null periods or collapse them with other phases. The maps were normalized (based on a scale of 100 units) so that the length of each phase represents its proportion of the original data. Normalization is helpful when comparing phase maps for phenomena that have different lengths of event sequences.

Fig. 7 and Fig. 8 show the phase maps for the two systems. The phase map for the Financial Sales Reporting System reflects two distinct periods of change in the system (period 1 occurs between 10 and 30 on the scaled timeline and period 2 occurs between 45 and 80 on the scaled timeline). In both periods, significant addition of new modules to the systems occurred. The first period appears to be characterized largely by addition of new modules and corrections until the modules stabilize. The second period involves more complex enhancements as well as addition of new modules and does not appear to have many phases of corrections. A number of phases relate to enhancements and corrections to computations or algorithms, suggesting

5. *Winphaser* is a software package for sequence analysis that was developed by Dr. Michael E. Holme, University of Utah.

Ordered Change Events

Financial Sales Reporting System	Manifest Shipping System
NewProgram	NewProgram
NewProgram	CorrectDat
NewProgram	CorrectLog
NewProgram	EnhDatChg
NewProgram	EnhDatChg
NewProgram	EnhDatChg
NewProgram	EnhDatChg
NewProgram	EnhLogChg
NewProgram	EnhLogChg
EnhLogAdd	NewProgram
EnhDatAdd	NewProgram
EnhLogAdd	NewProgram
NewProgram	NewProgram
CorrectLog	NewProgram
CorrectDat	NewProgram
CorrectDat	NewProgram
CorrectLog	NewProgram
CorrectLog	NewProgram
NewProgram	NewProgram
CorrectLog	NewProgram
.	.
.	.
.	.
n=841 total change events	n=652 total change events

Fig. 6. First 20 ordered change events for two systems.

that this system may be computationally complex. The phase map for the Manifest Shipping System on the other hand, reflects changes that do not appear to coalesce into a small number of distinct periods. Frequent enhancements to the user interface suggest that this system may be more interactive than the other system.

2.10 Gamma Analysis

While intriguing, these phase maps provide only a starting point for understanding and comparing the evolution of the systems. A more precise, statistical analysis of the phases is needed. Such an analysis can be accomplished using gamma analysis. Gamma analysis [34] provides a measure of the general order of phases in a sequence and a measure of the distinctness or overlap of phases. It calculates a gamma score, a nonparametric statistic based on the Goodman-Kruskal gamma that assesses the proportion of "A" phases that precede or follow "B" phases in a sequence. A pairwise gamma score is given by $(P - Q)/(P + Q)$, where P is the count of A phases preceding B phases, and Q is the count of B phases preceding A phases. Gamma analysis of a sequence yields a table of pairwise gamma scores for each possible pair of event types. The gamma tables for the Financial Sales Reporting System and the Manifest Shipping System are shown in Table 5 and Table 6. Note that the gamma tables are approximately symmetric; the northeast corner is the inverse of the southwest corner of the matrix.

The precedence score for a phase is the average of its pairwise gamma scores. The precedence score indicates the location of the phase in the overall ordering of phases and can range from -1 to $+1$. A precedence score close to -1 suggests that the phase occurs toward the end of the sequence, while a score close to $+1$ indicates that the phase occurs toward the beginning of the sequence. The separation score between a pair of phases can be determined by the absolute value of the pair-wise gamma scores. It assesses the relative distinctness of the phases and can range from 0 to 1. The closer the score is to 1, the greater the separation of phases. A value of gamma near 0 indicates that the phases do not precede or follow each other systematically. Typically a value of 0.50 or greater is used as a cutoff to indicate that the phases are separate [34]. This value can be interpreted to mean that the phases are distinct, and one phase precedes or follows the other at least 50 percent of the time.

For example, referencing Table 5 and Table 6, the precedence scores of -0.87 and -1.00 for the Enhance Data Handling Delete Phase (*EnhDatDel*) in the Financial Sales Reporting System and Manifest Shipping System, respectively, indicate that this phase occurs near the end relative to the other phases for both systems. The separation scores for the *EnhDatDel* phase (1.00 in both systems) indicate that this phase is distinct from the other phases.

2.11 Gamma Mapping

Gamma mapping is the final process in gamma analysis. Precedence and gamma separation scores are used to construct gamma maps. Phases are ordered sequentially on the basis of precedence scores (ordered from $+1$ to -1). Boxes are drawn to indicate the degree of separation of the phases. Phases with a separation score greater than 0.50 are boxed. Those with separation scores between 0.50 and 0.25 are indicated with an incomplete box. Those with a separation score below 0.25 are not boxed separately, but rather together to indicate the high degree of overlap. The gamma maps provide a concise way to compare phase typologies.

Fig. 9 and Fig. 10 show the gamma maps for the Financial Sales Reporting System and Manifest Shipping System. These figures suggest that while there are some similarities in the evolutionary phase typologies for the systems, there

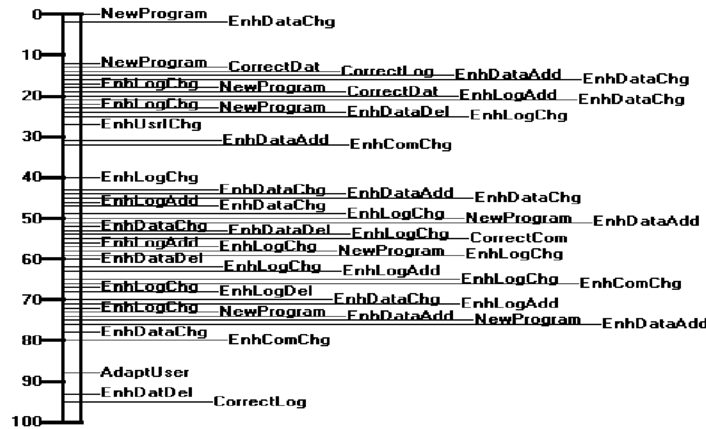

Fig. 7. Financial sales reporting system phase map.

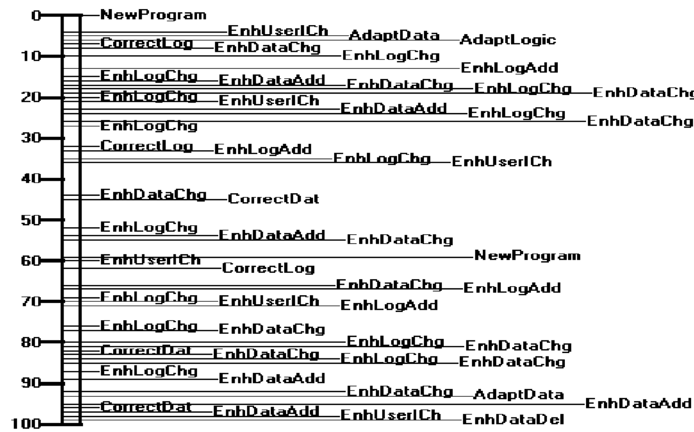

Fig. 8. Manifest shipping system phase map.

are also significant differences. The gamma map for the Financial Sales Reporting System begins with a large phase in which new modules are introduced, and changes are made to data handling, logic, and user interfaces. The next phase includes additions and deletions of data handling and changes to computations. This phase is followed by separate phases for deletion of logic, corrections to logic, and adaptations to the user interface. The final phase in the typology includes deletions of data handling.

The evolutionary phase typology for the Manifest Shipping System begins with the phase of new module introduction followed by logic adaptations. The next large phase includes first a subphase in which logic and user interfaces are changed and corrected. This subphase is followed by a subphase that focuses on adds, changes, and corrections to data handling. The final phase in the typology focuses on deletions of data handling.

Comparing the typologies for the systems, it appears that both systems start and end with rather similar phases. Generally both systems begin with introduction of new modules and changes to logic and user interfaces. Both systems appear to have later phases in which logic or data handling are deleted. However, the intervening phases are quite different for the systems. Further analysis could examine characteristics of the systems that may explain differences in their evolutionary paths. These characteristics could include business and environmental factors as well as technical elements of the systems.

To illustrate, one could relate the size, complexity and cost characteristics of the systems to the event frequency, phase maps, and gamma maps and use them to make a number of observations and propositions. The Financial Sales Reporting system has higher complexity in terms of data and computations relative to the Manifest system (as reflected in the normalized complexity metrics). Such complexity may be correlated with the higher frequency of corrections, adaptations, and enhancements to data handling and computations for the Financial system. The Manifest Shipping system is more interactive as reflected by the high percent of on-line (62 percent) vs. batch (38 percent) modules. This may relate to the higher frequency with which the user interface is modified in this system. Thus, one could predict evolution patterns based upon the design characteristics and functionality embodied in the systems. One could also formulate hypotheses to predict the consequences of following different evolution typologies. For our systems, the gamma map for the Financial system appears to be more disorganized and less principled (the

TABLE 5
Gamma Table*

FINANCIAL SALES REPORTING SYSTEM														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. NewProgram	0.00	0.13	0.22	-0.78	-0.43	-0.24	-0.33	-0.39	-0.11	-0.56	-0.33	-0.56	-1.00	-1.00
2. EnhDatChg	-0.13	0.00	-0.05	-0.83	-0.50	-0.46	-0.57	-0.58	-0.20	-0.61	-0.70	-0.70	-1.00	-1.00
3. CorrectDat	-0.22	0.05	0.00	-0.83	-0.86	-0.93	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
4. CorrectLog	0.78	0.83	0.83	0.00	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.67
5. EnhDatAdd	0.43	0.50	0.86	-0.67	0.00	0.22	0.05	0.07	0.71	-0.28	-0.14	-0.14	-1.00	-1.00
6. EnhLogChg	0.24	0.46	0.93	-0.67	-0.22	0.00	-0.22	-0.07	0.47	-0.29	-0.20	-0.87	-1.00	-1.00
7. EnhLogAdd	0.33	0.57	1.00	-0.67	-0.05	0.22	0.00	0.25	0.67	-0.20	0.33	-0.67	-1.00	-1.00
8. EnhDatDel	0.39	0.58	1.00	-0.67	-0.07	0.07	-0.25	0.00	0.50	-0.29	0.00	-1.00	-1.00	-1.00
9. EnhUsrlChg	0.11	0.20	1.00	-0.67	-0.71	-0.47	-0.67	-0.50	0.00	-1.00	-1.00	-1.00	-1.00	-1.00
10. EnhComChg	0.56	0.61	1.00	-0.67	0.28	0.29	0.20	0.29	1.00	0.00	0.06	-0.06	-1.00	-1.00
11. CorrectCom	0.33	0.70	1.00	-0.67	0.14	0.20	-0.33	0.00	1.00	-0.06	0.00	-1.00	-1.00	-1.00
12. EnhLogDel	0.56	0.70	1.00	-0.67	0.14	0.87	0.67	1.00	1.00	0.06	1.00	0.00	-1.00	-1.00
13. AdaptUser	1.00	1.00	1.00	-0.67	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	-1.00
14. EnhDatDel	1.00	1.00	1.00	-0.67	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00
Precedence Scores	0.41	0.56	0.83	-0.70	0.03	0.19	0.02	0.14	0.52	0.12	0.05	-0.33	-0.72	-0.87

**For both Table 5 and Table 6, the column numbers match the row numbers. The text labels (e.g., "NewProgram") are not included due to space limitations.*

TABLE 6
Gamma Table

MANIFEST SHIPPING SYSTEM											
	1	2	3	4	5	6	7	8	9	10	11
1. NewProgram	0.00	-0.71	-0.87	-0.60	-0.87	-0.79	-0.71	-0.86	-0.85	-0.69	-1.00
2. EnhUserlChg	0.71	0.00	-0.29	0.87	-0.24	-0.31	0.18	-0.26	-0.42	-0.56	-1.00
3. AdaptData	0.87	0.29	0.00	0.33	0.33	0.33	0.33	0.33	0.00	0.19	-1.00
4. AdaptLogic	0.60	-0.87	-0.33	0.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00	-1.00
5. CorrectLog	0.87	0.24	-0.33	1.00	0.00	-0.21	0.17	-0.46	-0.42	0.04	-1.00
6. EnhDatChg	0.79	0.31	-0.33	1.00	0.21	0.00	0.27	0.13	-0.37	0.11	-1.00
7. EnhLogChg	0.71	-0.18	-0.33	1.00	-0.17	-0.27	0.00	-0.23	-0.55	-0.40	-1.00
8. EnhLogAdd	0.86	0.26	-0.33	1.00	0.46	-0.13	0.23	0.00	-0.43	-0.01	-1.00
9. EnhDatAdd	0.85	0.42	0.00	1.00	0.42	0.37	0.55	0.43	0.00	0.33	-1.00
10. CorrectDat	0.69	0.56	-0.19	1.00	-0.04	-0.11	0.40	0.01	-0.33	0.00	-1.00
11. EnhDatDel	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00
Precedence Scores	0.79	0.13	-0.20	0.76	0.01	-0.11	0.14	-0.09	-0.34	-0.10	-1.00

phases are not as separate as in the Manifest system and do not coalesce into distinct and coherent subphases). This may suggest entropy in the Financial system. Note that the average costs to enhance, correct, and adapt this system are extraordinarily high relative to the Manifest system despite the similarities in size and functionality (e.g., it costs almost $6,000 to make one correction to the Financial system, vs. less than $400 per correction for the Manifest system). One could, therefore, predict that evolution patterns reflecting disorganized activity suggest system entropy, and correlate this with high maintenance costs.

3 Research Approach Conclusions

3.1 Discussion

Given that this is an article about research methods, it seems appropriate to focus the discussion on the research approach taken on this project. Given that empirical

Fig. 9. Gamma map for financial sales reporting system.

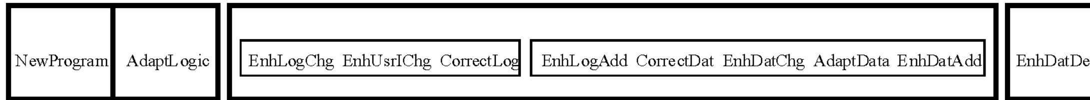

Fig. 10. Gamma map for manifest shipping system.

research in software evolution is so rarely accomplished, in hindsight, what are seen as the critical success factors and methodological issues here?

Most important for the success of empirical research on software evolution is the participation of a good commercial partner. The minimum characteristics of a good data site for this type of research must include a site that retains a significant number of past versions or releases of the software systems being studied. Unfortunately, since most organizations do not require these previous versions to conduct day-to-day operations, they are typically discarded, which means that their value as artifacts in researching and improving the organization's software process cannot be tapped. Second, and related to the first, not only did this organization have the prior system versions, but the prior versions were meaningful, because the organization had enforced good change control discipline. Modifications to systems were consistently logged. This allowed the research team to recreate the system's history. Finally, and an overarching criterion, the organization was highly cooperative and generous with the research team, sharing not only their data, but also staff time and other resources. Without this cooperation, the research would not have been possible.

Another whole category of critical success factors includes the recognition that, in order to conduct an empirical research program of this magnitude, a highly disciplined research approach is required. The significantly larger scale of the data (25,000 change events vs. typical 25 releases in prior research) meant that more typical, relatively ad hoc approaches to such procedures as data collection and data analysis, would not be effective. In addition, the scale of the project meant that a relatively larger research team would need to be assembled in order to provide both the manpower and the variety of skills necessary to accomplish the work. Of course, such an approach is significantly more costly than is usual. In addition, having a research team implies that attention must be paid to a variety of coordination tasks: formal procedures, documentation, training, reliability testing, and database management, to name the most important.

In addition to performing original longitudinal research in software evolution, we also had the desire to expand upon prior software maintenance research, e.g., having an error classification scheme with 30 categories rather than only the typical three (corrective, adaptive, and perfective). Such an approach, while desirable from the standpoint of improving the richness of the data and allowing for more detailed examination of the process, adds significantly to the time and cost of the work.

We had both a willingness and a need to consider analytical approaches from other research domains and disciplines. The need arose from the relatively limited prior empirical research in software evolution, which does not provide a broad base for this work. As a consequence, many new choices on research methods had to be made. In this article, we have illustrated one such new method, phasic analysis from the social sciences. We have found the social sciences to be relatively advanced in the examination of historical patterns and processes and believe many of the methods developed in these disciplines can be gainfully applied in analyzing the evolution of software systems.

3.2 Summary

In this paper, we have focused on the processes, design, and structure of our empirical research into software evolution. We began by reviewing the prior research of software evolution. Our review revealed a limited number of studies that have been largely focused in two major areas: understanding and describing the dynamics of software evolution and developing a taxonomy of maintenance categories. These studies have revealed initial insights into software evolution, suggesting that there may be recognizable patterns in the evolution of software systems, and that such patterns may be associated with system entropy and other outcomes. While intriguing, the scope of these studies has been limited in terms of research approach and empirical data. We argue that there is a need for more longitudinal research of software maintenance. Many of the problems in software maintenance are caused by a lack of precise knowledge of the maintenance process and of the cause and effect relationships between software practices and maintenance outcomes. The major benefits and penalties of software practices for maintenance are realized over the life cycle of the software, a life cycle that typically extends to many years and involves many different kinds of maintenance activities. An understanding of how software maintenance activities and costs change over time can inform current maintenance and development practice. Longitudinal empirical studies of software evolution are essential in providing the data necessary to perform such

analyses and develop new software engineering principles and theories.

We have designed an approach for longitudinal research that enlarges the scope of the empirical data available on software evolution so that evolution patterns could be examined across multiple levels of analysis (system and module), over longer periods of time, and could be linked to a number of organizational and software engineering factors. Our study has involved the construction of a very large database chronicling, in considerable detail, the historical growth and change of 23 software applications over a period of up to 20 years. In order to accomplish this we first identified and partnered with a commercial organization with both sufficient raw data and a willingness to cooperate with the research. We designed and implemented a set of procedures to ensure that the data were collected, transformed, and analyzed in a highly reliable manner, so that the final results will have high credibility. In addition, we have created and adapted new methods and techniques with which to deal with these data, including the categorization of maintenance causes into a range of parameters, and the application of phasic analysis from the behavioral sciences. We believe that this approach has created a rich and reliable empirical data set from which a variety of research results will be driven, beginning with evaluation of existing scientific theories (e.g., the "laws" of software evolution), and ending with the development of completely new insights made possible by the novel data.

Other studies of software evolution could build upon and extend our approach by examining software evolution in different contexts. It would be interesting, for example, to compare the evolution of traditional "legacy" systems with systems that have been developed using different approaches such as object-oriented design. Such a comparison could be instructive in determining the efficacy of improved software design methods. Furthermore, it would be informative to compare evolution patterns across different kinds of organizations and industries to assess the importance of environmental factors in driving software evolution. Finally, it may be useful to examine the historical empirical data we have collected by employing other methods such as simulation to *predict* the occurrence of evolution patterns in software systems.

Acknowledgments

Research support from the University of Pittsburgh's Institute for Industrial Competitiveness and from Carnegie Mellon University is gratefully acknowledged. Helpful comments were received from the editors and two anonymous referees.

References

[1] A. Abbott, "Transcending General Linear Reality," *Sociological Theory,* vol. 6, pp. 169–86, 1988.

[2] A. Abbott, "A Primer on Sequence Methods," *Organization Science,* vol. 1, pp. 375–392, 1990.

[3] A. Abbott, "Sequence Analysis: New Methods for Old Ideas," *Ann. Review of Sociology,* vol. 21, pp. 23–40, 1995.

[4] A. Abbott and A. Hyrcak, "Measuring Resemblance in Sequence Data: An Optimal Matching Analysis of Musicians' Careers," *Am. J. Sociology,* vol. 96, pp. 144–85, 1990.

[5] R.D. Banker, H. Chang, and C.F. Kemerer, "Evidence on Economies of Scale in Software Development," *Information and Software Technology,* vol. 36, no. 5, pp. 275–282, 1994.

[6] R.D. Banker and C.F. Kemerer, "Scale Economies in New Software Development," *IEEE Trans. Software Eng.,* vol. 15, no. 10, pp. 1,199–1,205, 1989.

[7] R.D. Banker and S. Slaughter, "Field Study of Scale Economies in Software Maintenance," *Management Science,* vol. 43, no. 12, pp. 1,709–1,725, 1997.

[8] V.R. Basili, L.C. Briand, S. Condon, Y.-M. Kim, W. Melo, and J. Valett, "Understanding and Predicting the Process of Software Maintenance Releases," *Proc. 18th Int'l Conf. Software Eng.,* Berlin, 1996.

[9] L.A. Belady and M.M. Lehman, "A Model of Large Program Development," *IBM Systems J.,* vol. 15, no. 1, pp. 225–252, 1976.

[10] K. Bennett, "Software Evolution: Past Present and Future," *Information and Software Technology,* vol. 38, pp. 673–680, 1996.

[11] G. Box, G. Jenkins, and G. Reinsel, *Time Series Analysis: Forecasting and Control.* Englewood Cliffs, N.J.: Prentice Hall, 1994.

[12] L.C. Briand and V.R. Basili, "A Classification Procedure for the Effective Management of Changes during the Maintenance Process," *Proc. Conf. Software Maintenance,* 1992.

[13] A. Bryk and S. Raudenbush, *Hierarchical Linear Models: Applications and Data Analysis Methods.* Newbury Park, Calif.: Sage Publications, 1992.

[14] S. Cha, I.S. Chung, and Y.R. Kwon, "Complexity Measures for Concurrent Programs Based on Information-Theoretic Metrics," *Information Processing Letters,* vol. 46, pp. 43–50, 1993.

[15] C.K.S. Chong Hok Yuen, "An Empirical Approach to the Study of Errors in Large Software under Maintenance," *Proc. Second Conf. Software Maintenance,* IEEE, Washington, DC., 1985.

[16] C.K.S. Chong Hok Yuen, "A Statistical Rationale for Evolution Dynamics Concepts," *Proc. Conf. Software Maintenance,* Austin, Tex., 1987.

[17] C.K.S. Chong Hok Yuen, "On Analyzing Maintenance Process Data at the Global and Detailed Levels: A Case Study," *Proc. Fourth Conf. Software Maintenance,* IEEE, Phoenix, Az., 1988.

[18] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement,* vol. 20, no. 1, pp. 37–46, 1960.

[19] C.R. Cook and A. Roesch, "Real-Time Software Metrics," *J. Systems and Software,* vol. 24, no. 3, pp. 223–237, 1994.

[20] J. Dvorak, "Conceptual Entropy and its Effect on Class Hierarchies," *Computer,* pp. 59–63, 1994.

[21] B.A. Fisher, "Decision Emergence: Phases in Group Decision Making," *Speech Monographs,* vol. 37, pp. 53–66, 1970.

[22] D. Gefen and S.L. Schneberger, "The Non-Homogeneous Maintenance Periods: A Case Study of Software Modifications," *Proc. Conf. Software Maintenance,* IEEE, Monterey, Calif., 1996.

[23] W.H. Green, *Econometric Analysis,* second ed., New York: Macmillan, 1993.

[24] W. Harrison, "An Entropy-Based Measure of Software Complexity," *IEEE Trans. Software Eng.,* vol. 18, no. 11, pp. 1,025–1,029, Nov. 1992.

[25] M.E. Holmes and R.E. Sykes, "A Test of the Fit of Gullivers's Phase Model to Hostage Negotiations," *Comm. Studies,* vol. 44, pp. 38–55, 1993.

[26] IEEE, *IEEE Standard for Software Maintenance.* pp. 39, New York: IEEE, 1993.

[27] M. Keil, "Pulling the Plug: Software Project Management and the Problem of Project Escalation," *MIS Quarterly,* vol. 19, no. 4, pp. 421–448, 1995.

[28] C.F. Kemerer, "Empirical Research on Software Complexity and Software Maintenance," *Annals of Software Eng.,* vol. 1, no. 1, pp. 1–22, 1995.

[29] C.F. Kemerer and S. Slaughter, "Need for More Longitudinal Studies of Software Maintenance," *Proc. Int'l Workshop Empirical Studies of Software Maintenance,* Monterey, Calif., 1996.

[30] C.F. Kemerer and S. Slaughter, "Methodologies for Performing Empirical Studies: Report from the International Workshop Empirical Studies of Software Maintenance," *Empirical Software Eng.,* vol. 2, no. 2, 1997.

[31] R.J. Landis and G.G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics,* vol. 22, pp. 159-174, 1977.

[32] M.M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution," *Proc. Special Issue Software Eng.*, IEEE, vol. 68, no. 9, pp. 1,060–1,076, 1980.
[33] M.M. Lehman, J.F. Ramil, P.D. Wernick, D.E. Perry, and W.M. Turski, "Metrics and Laws of Software Evolution—The Nineties View," *Proc. Fourth Int'l Software Metrics Symp., Metrics '97*, Albuquerque, N.M., 1997.
[34] D.C. Pelz, "Innovation Complexity and the Sequence of Innovating Stages," *Knowledge: Creation, Diffusion, Utilization*, vol. 6, pp. 261-291, 1985.
[35] D.E. Perry, "Dimensions of Software Evolution," *Proc. Conf. Software Maintenance*, IEEE, 1994.
[36] M.S. Poole and J. Roth, "Decision Development in Small Groups IV: A Typology of Group Decision Paths," *Human Comm. Research*, vol. 15, no. 3, pp. 323-356, 1989.
[37] H.D. Rombach, B. Ulery, and J.D. Valett, "Toward Full Life Cycle Control: Adding Maintenance Measurement to the SEL," *J. Systems and Software*, vol. 18, pp. 125-138, 1992.
[38] R. Saberwhal and D. Robey, "An Empirical Taxonomy of Implementation Processes Based on Sequences of Events in Information System Development," *Organization Science*, vol. 4, pp. 548-576, 1993.
[39] N.F. Schneidewind, "The State of Software Maintenance," *IEEE Trans. Software Eng.*, vol. 13, no. 3, pp. 303–310, Mar. 1987.
[40] B.A. Semic and D.J. Cnary, "Trait Argumentativeness, Verbal Aggressiveness, and Minimally Rational Argument: An Observational Analysis of Friendship Discussions," *Comm. Quarterly*, vol. 45, no. 4, pp. 355–378, 1997.
[41] J.R. Sparks, C.S. Areni, and K.C. Cox, "An Investigation of the Effects of Language Style and Communication Modality on Persuasion," *Comm. Monographs*, vol. 65, no. 2, pp. 108–125, 1998.
[42] A.M. Stillwell and R.F. Baumeister, "The Construction of Victim and Perpetrator Memories: Accuracy and Distortion in Role-Based Accounts," *Personality and Social Psychology Bulletin*, vol. 23, no. 11, pp. 1,157–1,172, 1997.
[43] E.B. Swanson, "The Dimensions of Maintenance," *Proc. Second Int'l Conf. Software Eng.*, 1976.
[44] E.B. Swanson and C.M. Beath, "Departmentalization in Software Development and Maintenance," *Comm. ACM*, vol. 33, no. 6, pp. 658–667, 1990.

Chris F. Kemerer received the BS degree in decision sciences and economics from the Wharton School at the University of Pennsylvania and the PhD degree from the Graduate School of Industrial Administration at Carnegie Mellon University. He is David M. Roderick, chair in Information Systems at the Katz Graduate School of Business, University of Pittsburgh. Previously, he was an associate professor at MIT's Sloan School of Management. His research interests are in the measurement and modeling of software development for improved performance; he has previously published articles on these topics in *Communications of the ACM*, *Computer*, *IEEE Software*, *IEEE Transactions on Software Engineering*, *Information and Software Technology*, *Information Systems Research*, *Management Science*, *Sloan Management Review*, and others. Dr. Kemerer serves, or has served, on the editorial boards of the *Communications of the ACM*, *Information Systems Research*, the *Journal of Organizational Computing*, the *Journal of Software Quality*, and *MIS Quarterly*. He is currently the departmental editor for information systems at *Management Science*.

Sandra Slaughter received her doctorate degree in information systems from the University of Minnesota and an MBA degree in finance from Indiana University. She is an assistant professor in the Graduate School of Industrial Administration at Carnegie Mellon University. Dr. Slaughter has worked in the information systems industry for 10 years in information systems planning, project management, and systems analysis and maintenance. Her research is focused on productivity and quality improvement in the development and maintenance of information systems and on effective management of information technology professionals. Her research is field-based and empirical, involving collection and analysis of data on software complexity, project characteristics, systems environment, and project team factors in order to estimate the impact of managerial and technological variables on software productivity and quality. Her dissertation on software development practices and software maintenance performance received the ICIS Best Dissertation Award in 1995. She has published articles in leading research journals and conferences in management, information systems, and management science.