Understanding Software Maintenance Work

SALAH BENDIFALLAH AND WALT SCACCHI, MEMBER, IEEE

Abstract—**Software maintenance can be successfully accomplished if the computing arrangements of the people doing the maintenance are compatible with their established patterns of work in the setting. To foster and achieve such compatibility requires an understanding of the reasons and the circumstances in which participants carry out maintenance activities. In particular, it requires an understanding of how software users and maintainers act toward the changing circumstances and unexpected events in their work situation that give rise to software system alterations. To contribute to such an understanding, we describe a comparative analysis of the work involved in maintaining and evolving text-processing systems in two academic computer science organizations. This analysis shows that how and why software systems are maintained depends on occupational and workplace contingencies, and vice versa.**

Index Terms—**Articulation work, computing milieux, maintenance work, primary work, social analysis of computing, software evolution, software maintenance, software productivity, text-processing.**

I. Introduction

SOFTWARE maintenance is complex and costly. Maintenance activities are estimated to take up more than half of the life-cycle cost of software systems [7], [25]. Yet software maintenance remains the least understood and most problematic part of the software process.

Reducing the cost of software maintenance entails understanding the various kinds of alterations that people make to software systems and providing tools for carrying out these alterations. But more importantly, we believe it entails understanding the circumstances that give rise to why alterations are made, how they are performed, and how these circumstances are tied to the evolution of the *work arrangements* in the setting: the organization and distribution of productive resources (computers, software tools, skills, time, money, computer facility staff, etc.) committed to supporting current work activities, the constraints upon use of these resources, and how people work within these constraints to transform the resources into finished products or services [11], [30], [10].

A common classification of alterations [40], [27], [13], [28], [1], [41] distinguishes *corrective, adaptive, perfective, and preventive* alterations according to their *immediate* causes, the evolution of system requirements or the inadequacy of the current system. Adaptive, perfective, and preventive activities are typically considered enhancement activities. They are the major share of maintenance work, consuming as much as 75 percent or more of maintenance time [25]. This is especially true in cases where maintenance is *an afterthought* and an explicitly separate phase of the software life-cycle, completely independent from development. The recognition of enhancement as the dominant maintenance activity has given rise to a new approach to software engineering, where incremental system enhancement is deemed the main software activity and hence tools and methods are devised which reclaim maintenance into the realm of development activities [2], [33], [3].

However, further emphasis on tools cannot alone solve the problems of software maintenance. Merely mobilizing technical means cannot suffice to solve what appear to be more fundamental problems concerning the reasons systems are altered and the conditions under which they are altered. Maintenance activities are labor intensive and involve programming as well as nonprogramming tasks. Enhancement activities, in particular, depend on work arrangements in the organizational setting. Thus, following the introduction of the new enhancement tools, demands on work arrangements must be shifted for assuring that these tools can be effectively utilized, in particular for assuring that such tools can be successfully *fit* into the routine work practices of the setting [30].

Our purpose in this paper is to contribute to a better understanding of the fundamental aspects of software maintenance *work*. Specifically, we want to understand *the ways local circumstances in the workplace affect how and why people perform software maintenance tasks*, and conversely, *how maintenance work affects workplace arrangements*. Local circumstances include the incentives and constraints for *why* people alter their software systems, and indicate *when* people act to maintain their systems. The workplace specifies *where* maintenance work is performed and the ways it is organized. *How* people order and perform their maintenance work also entails *who* does this work, and *what* kind of maintenance activity is performed.

We present an empirical analysis of two cases of comparable software systems in similar organizations. The case studies concern the evolution of text-processing systems in two academic computer science organizations, CSD and CSRO.[1] CSD [5], [6] is the computer science department of a major university, and CSRO [17], [30], [18] is the computer science research organization at-

Manuscript received September 2, 1985; revised February 11, 1986. This work was supported by AT&T Information Systems, TRW Defense Systems Group, and IBM through Project Socrates at USC.

The authors are with the Department of Computer Science, University of Southern California, Los Angeles, CA 90089.

IEEE Log Number 8610901.

[1]We use pseudonyms throughout.

tached to another major university. Our choice to examine the evolution of similar software systems in similar settings is an attempt to mitigate the influence of many potentially confounding variables.

We use the CSD and CSRO cases to illustrate the kind of analytical detail needed to understand why and how software maintenance work is accomplished. These two cases alone cannot represent how software maintenance work is performed in every organization. However, our research design (presented in Appendix) provides an analytical framework for generalizing our findings and assessing findings from related research. Subsequently, insofar as any model or systematic account of software maintenance work can be assessed in terms of our research design, it should also be able to account for the kind of activities and situations we describe.

The thrust of our analysis is to explicitly consider the work situation in which a software system evolves. Rather than examining maintenance work by focusing on the features of the system itself, we explicitly consider the coevolution of the participants' work tasks with their local work arrangements.

We start our investigation in Section II by examining what the people who use software systems do as their *primary work*. Their involvement in particular tasks typically matches either some current occupational or career interest, or some circumstantial commitment which must be met in order to pursue other work activities. Primary work includes all the tasks for which a person is explicitly responsible and rewarded. Ideally, these tasks are in line with each person's interests in the organization (e.g., professional advancement) and are tasks each person would rather concentrate on if given full discretion. At CSD and CSRO, most participants are computing specialists whose primary work includes computer science research work. Research publications and related technical documents are an important, professionally recognized product of this kind of work (cf. [19]). The use of a text-processing system is central to the production of these reports.[2]

We examine how CSD and CSRO participants use the text-processing systems to accomplish their primary work. Each text-processing task at CSD and CSRO has some relationship to tasks performed or controlled by other participants. At any time, the successful coordination of these interlocking tasks depends on a variety of social and technical arrangements. The work required to coordinate and align these arrangements to accomplish the tasks at hand is called *articulation work* [39], [10], [37], [9]. This is the focus of our investigation in Section III. In particular, we investigate the participants' activities when their primary work tasks get *dis*-articulated. In both settings, different kinds of *re*-articulation work emerge in response to unexpected breakdowns in the organization of primary work. Participants choose one of two alternatives. The first is to take the system as a *fait accompli* and *accommodate* the way they work to the way the system operates within local computing arrangements. The other is to make changes in the system as well as in the work arrangements, i.e., negotiate the appropriate maintenance alteration to be performed and who will carry out the work. Consequently, *how and why software maintenance work is performed depends on how the related articulation work is accomplished.*

We develop this conclusion in Section IV by examining the similarities and differences between the two cases. The evolutionary courses of the two systems diverge. At CSRO, the text-processing system evolves into multiple user-personalized configurations, and all maintenance work on the system is performed by the users themselves in a loosely coupled manner. At CSD, the text-processing system evolves so that a dominant configuration emerges. The original user/maintainer of this configuration develops a reputation of expertise, and maintenance work on the system thereby becomes this person's primary work through an opportune career option. For us, the divergence between the two evolutionary courses indicates a relationship of *mutual influence* between the circumstances of maintenance work, the participants' strategies for prioritizing the demands of their work, and the incentives for and constraints on these demands.

Last, in Section V, we discuss related research and then conclude in Section VI with a summary of our findings and their implications for understanding software maintenance work.

II. Primary Work

We first examine a primary work activity at CSD and CSRO, the production of research publications. Next, we survey the work arrangements which supported this activity. Then, we examine the text-processing tasks which CSD and CSRO participants performed and coordinated in carrying out this activity.

A. Primary Work at CSD and CSRO

Users of the text-processing systems at CSD and CSRO were typical of those found in academic computer science organizations: faculty, project managers, research associates, graduate assistants, systems support staff, and administrative and clerical staff. At the outset of each academic term, faculty, graduate students, and research associates might join or leave. Systems support staff as well as members of the managerial, administrative, and clerical staff could, on the other hand, leave or join at any time.

At CSD and CSRO, the process of producing a research publication emerged from the shared construction of an idea or alternate work arrangement (hereafter, *concept*) deemed a departure from practices described in related publications (cf. [19]). The concept would pass through several stages of development. From stage to stage, it took on forms such as conversational conjectures, informal debates and clarifications, notes, memoranda, overhead projection transparencies, group presentations, preliminary

[2]In fact, regular use of most computing applications leads to the production of reports, listings, and formatted displays of one kind or another.

drafts, and polished technical reports. In addition, some researchers constructed software systems and related documentation in pursuing their concept, often to a point where their software development work would dominate the effort they would commit to producing research publications.[3] At any stage, any number of participants could (un)knowingly collaborate in the process of constructing the publication. Similarly, at any stage, the process could be put off when unexpected circumstances arose that sidetracked or derailed the participants' interest in the concept.

The actual manner in which a concept was investigated was bound to the career contingencies of the computing participants involved. A junior researcher could adopt a concept and work on it to solve a particular problem at hand, often in connection with an ongoing project and a senior researcher. For example, a graduate student decided to develop a knowledge-based program explanation system to further his interest in advancing knowledge-based systems technology, as part of the reknown FOO project. In such cases, the concept might evolve no farther than an internal report, technical memoranda, or other related artifacts (e.g., a concept demonstration system) unless the researchers believed their research findings were substantial enough to further develop, publish, and circulate. What followed was an explicit collaboration and commitment between the researchers to "get the machine out the door" (cf. [14]) by producing a professional-quality report that could be disseminated to colleagues.

Initial dissemination took place among local research and discussion groups, where the potential publication was circulated, critiqued, debated, and revised. Researchers could utilize available networks (social, professional, and electronic) to announce an emergent publication "in press" or "in preparation," or otherwise bring the attention of prospective readers to the emergence of the publication. Public distribution occurred when the publication was circulated through the appropriate marketplace of ideas as a technical report, an article in a professional journal, or a monograph. This marketplace included other academic organizations where participants strove to stay abreast of research developments by colleagues pursuing similar research and publications.

When a publication was distributed and cited in the professional literature and among cohorts, the resulting recognition enabled new professional opportunities for the publication's authors, and contributed to furthering their reputations and that of their organization. These outcomes carved out a rather pivotal role for the text-processing systems used at CSD and CSRO. These systems encompassed the computational resources used to produce intermediate and final versions of emerging professional publications. We describe them next as we review the work arrangements in each setting.

B. The Work Arrangements and Text-Processing Systems

We first examine the incentives which sustained the participant's primary work at CSD and CSRO, then describe the computing infrastructure which both facilitated and constrained accomplishment of this work.

Participants' Incentives: The focal incentives of most participants at CSD and CSRO were shaped by shared commitments to create and publish valuable research results. These commitments were renewed through the actions of individual researchers, collegial reference groups, academic units/universities, professional associations, and research funding agencies. Through carefully prepared and revised publications of research results, participants could:

- experience the self-satisfying accomplishment of a job well done, a new entry in their curriculum vita, and the pride in personally distributing copies to friends and colleagues;
- increase their professional status and reputation when recognized as making a significant contribution, and in turn, increase their influence in their academic community;
- establish or reinforce their identity with an invisible college of scholarly cohorts via cocitation and joint authorship;
- help gain promotion, salary increases, and a larger share of resources allocated within their academic unit, or job offers elsewhere;
- achieve widespread reproduction of the publication, increase their professional visibility, make public conference presentations, receive ceremonial awards (for special accomplishments), and achieve bibliographic archival all through the sustaining publication activities of professional associations (e.g., ACM, IEEE); and
- establish and maintain a conduit for resources flowing from funding agencies that in turn were acknowledged for their support and stimulation of the work leading to research publications.

Accordingly, participants regularly assessed which ensemble of these outcomes motivated their effort to produce a research publication.

Computing Facilities: CSD's and CSRO's participants used local computing resources in a manner commensurate with their level of research funding and their position in the organization. Not all users had computer terminals and high-quality printers readily accessible, nor did they all have the same allocation of system resources such as computing cycles and on-line disk storage space.

At CSD, all participants had free use of local computing facilities managed by the department. There was a terminal in an office for each faculty member and systems

[3]While participants at both CSRO and CSD might normally develop software systems as part of their primary work, at the time of our studies, researchers would more often receive professional rewards and promotions based on the publication of their concepts, rather than on only their demonstration of a concept by a software system.

programmer, and a shared terminal in each office for graduate students or research associates. Additional computing resources were available elsewhere within the university on a "pay as you go" basis to those who could afford it (mostly faculty with research grants and their research assistants). One such resource was a large time-shared system dedicated to text-processing, which we call the "pay" machine, in contrast to CSD's own "free" machine.

The communication facilities available to CSD participants for automatically distributing their documents or messages included three major electronic mail components. ZIP, the local mail system and NZIP, which handled nationwide network mail, were acquired from outside vendors. CEDEX, an integrated mail system, was later developed locally by Dr. T., a junior faculty member who wanted an integrated mail handling system. Because he sent and received mail on both the local area and the national computer network, he found separate mail burdensome and decided, after consulting with a few regular users, to unify the functions of ZIP and NZIP into a single software system.

At CSRO, computer terminals were provided by individual research projects according to thier funding arrangements; bigger projects had a larger staff and more terminals than smaller projects. Users in small or unfunded projects were often particularly pressed when many users in the large projects were also computing. Use of disk storage space and computing cycles on CSRO's single time-sharing machine was regulated by an allocation scheme enforced by a dedicated MONITOR program and an automatic file ARCHIVE facility.

Four other system components supported text-processing tasks at CSRO. NETMAIL, an integrated mail handling system, was acquired from another organization. BB, a bulletin board facility used as a repository of project communications and other public notices, was developed locally by users. The ARCHIVE utility was developed by local computer facility programmers to automatically store unused computer files (e.g., old documents or source program codes). Last, the MONITOR program was developed by the computer facility manager as a means to keep users from exceeding their allocated amounts of computer time or storage space and to invoke ARCHIVE whenever these amounts were exceeded.

These computing arrangements formed the backbone of the text-processing system in each setting. We now describe how the text-processing system components became part of each setting's work arrangements to support the participants' primary work, the production of research publications.

The Text-Processing Systems: The text-processing systems used at CSD and CSRO were distinguished mainly by the fact that they had evolved over different periods of time. Both systems had originally been developed outside of their setting of use. In both settings, the adoption of text-processing system components, communication components, and other supporting facilities was straightforward. Some parts came bundled with the computer system (from the system's manufacturer), while other parts were bought or brought in and adapted by eager users. At CSD, the text-processing system had been acquired from outside for local use and maintenance only one year prior to the time of study. In contrast, at the time of the study, some components of the text-processing system used at CSRO had a five-year history in the setting, during which they had undergone substantial in-house redevelopment for local use and maintenance.

The TEPS text-processing system we investigated at CSD had two major components: a screen-oriented text-editor (TEDS) and a text-formatter (TEFS). Both were developed at another academic computer science organization and were user-modifiable in order to fit into different patterns of use. TEDS was an extensible system, but had many idiosyncratic features which made it complex for some to learn. Installed on CSD's free machine in March 1984, TEDS had since been revised to mitigate its complexity. Nevertheless, CSD users still faced a tradeoff between its extensibility and its idiosyncratic features. TEFS was developed for computing users who produce technical academic manuscripts. It was first developed in 1978, and began to reach a broader community of users during 1979. By September 1984, TEFS was installed on CSD's free machine. Participants who had to access the pay machine primarily to use TEFS could then migrate to the free machine, with the prospect of continuing their established patterns of text-processing at a lower cost. Within a year, many CSD participants were using TEDS and TEFS on the free machine with sufficient frequency as to often bog down system performance during office hours.

The text-processing system components at CSRO were operational and in routine use at the onset of our inquiry in 1978. The most used components of the text-processing system included screen-oriented text-editors and the NEAT text-formatter. The text-processing components formed an eclectic ensemble of user-specific configurations developed locally in an *ad hoc* and fragmented manner. When the main computer system was installed at CSRO, it came with two line-oriented text-editors and one text-formatter supplied by the vendor. Different users imported screen-oriented text-editing, text-formatting, and other text-processing programs from other compatible computing facilities. The design history of these various text-processing tools was fragmented and hard to reconstruct. Also, there was no overall system design to coordinate the use of different system components. Consequently, the efforts to implement and modify the text-processing system at CSRO followed an unplanned trajectory based on individual participants' needs, interests, and dispositions.

The researchers' primary work with these text-processing systems at CSD and CSRO was typically organized into sequences of small tasks or *task chains* involving the computing resources available in the setting [30]. We examine task chains next.

C. Task Chains

A researcher's primary work with the text-processing system did not always require using the text-formatter. The text-editor alone would be used to produce documents intended for limited distribution within the working group of the author(s). Such documents required no more than an "acceptable" quality presentation. The text-formatter was usually used when it was necessary or desired to produce a document with "nice" quality presentation intended for outside distribution. The chains of text-processing tasks involved in a researcher's primary work mobilized a number of other computing resources, such as terminals, printers, and electronic mail systems. Depending on the current work arrangements, a text-processing task chain could materialize in different forms [30]. The following is an example of such a task chain:[4]

1) get access to a computer terminal;
2) gain access to the system;
3) create or alter text files with a text-editor;
4) format the text files;
5) obtain a formatted and printer-ready version of the files and verify that formatting/type setting is as desired;
6) gain access to a suitable printer and get the document printed;
7) verify that the formatting obtained is as expected; and
8) repeat any appropriate subchain until satisfaction or bottleneck.

Users of the text-processing systems at CSD and CSRO usually expected to perform and complete these tasks without any problem. However, should a system bug or any other interruption[5] arise in the current work arrangements, each user was responsible for bringing about whatever actions were necessary to carry out her/his task chain to completion. The more familiar the path down a task chain and the fewer the interruptions which *dis*-articulated the task chain, the less the amount of articulation work a user had to commit to doing in order to produce a document in the desired form. We examine this articulation work in the next section.

III. Articulation Work

In a routine excursion down a document-preparation task chain, users knew the various system operations and other actions to be performed. Otherwise, when unforeseen problems disarticulated the task chain, users consulted system documentation or other system users for cooperation in completing the task chain. We examine first how the successful completion of a primary work task chain gave rise to articulation activities and then examine the different forms of these articulation activities.

A. Task Chain Breakdowns Spawn Articulation Work

Completing a primary work task chain successfully is inherently nontrivial. Even though the task chain may be readily described and understood at some appropriate level of abstraction (e.g., our earlier description of a text-processing task chain), it is more accurately an *emergent* process. It is enacted somewhat differently in each particular work instance as the involved participant(s) may see fit to respond to contingencies in work arrangements [10].

At CSD and CSRO, whenever the excursion down a text-processing task chain became problematic—such as when the commitments of resources (including people) to some task chain were not met, when interruptions or bottlenecks were encountered—some articulation work had to be performed before the task chain could be resumed (either where it was stopped or as befitted the work arrangements resulting from the interruption). As the user of the text-processing system or the cooperative participants acted to resume the task chain, further problems could arise, leading to the emergence of more articulation work.

To understand the consequences of this emergent process for the eventual completion of a text-processing task chain and the subsequent effect on the user's primary work, let us consider a typical scenario of breakdown of a task chain and consequent emergence of articulation work at CSD. This scenario is summarized in Fig. 1, where the annotations in italics (corresponding to the underlined parts of the description) represent examples of articulation work.

Typically, a user's first response to a bottleneck was to try to accommodate to the new situation by initiating a quick remedial action. However, this response was not always successful. The user would then seek appropriate help from other nearby participants (e.g., another researcher, a member of the clerical staff, a member of the official maintenance staff). Getting help, however, required some negotiating since most participants were preoccupied with their own tasks. In the unfortunate circumstance where the user ended up being left to her own means, she would attempt to work around the difficulty. When this was not possible or not successful, the user would try some other means to work toward a solution by a reasonable deadline. In turn, this often resulted in postponing completion of the current task chain and switching to another, depending on the user's schedule and dispositions. Working around the problem often required further unexpected accommodation activities and sometimes even gave rise to some other (sub-)task chain. Moreover, getting someone to devote what could amount to a substantial effort to obtain a solution by a particular deadline required negotiation. If none of the above alternatives could be achieved, the user's last resort was to completely restructure the task chain—if possible, so as to bypass the

[4]Other examples of text-processing task chains can be found in [10], [18], [30].

[5]Such interruptions included running out of printer paper, jamming of the printer and consequent destruction of the current document copy, accidental deletion of the current document file, errant editor keystrokes that deleted emerging text, retrieving files that were automatically archived, encountering a high level of demand for shared computing resources, printer hardware failure, or an operating system crash.

Example: The printer paper box is empty on the evening before the research report must be submitted to meet a publication deadline.

Can I do something quickly about the bottleneck?
If so then do it and resume the (sub)task chain;
(*load more printer paper, then print the report*)
if not can I get immediate help from someone who knows what to do?
(*find someone who knows how to load more printer paper*)
if so then do it and resume the (sub)task chain;
(*print the report*)
if not can I afford to postpone work on this task chain?
(*can the report be delivered late without much hassle*)
if not then immediately attempt a work-around
(*the report must be sent out the next day by Express mail; so find another printer, move the document files to a backup system, or move to a typewriter*)
and resume the task chain;
if so can I get someone else to work on it by a reasonable deadline?
(*ask a graduate student to retrieve and load printer paper, and then make sure the printer is operating correctly*)
if so then postpone the rest of the task chain until the fix is made and switch to an alternate task chain;
(*e.g., update a bibliography file*)
if not can I fix it myself by some reasonable deadline?
(*procure enough printer paper and install it right away*)
if so then postpone the rest of the task chain until the fix is made;
if not attempt to restructure the task chain
(*send a letter explaining the delay*)
and propose to routinize the handling of the interruption via a software enhancement.
(*modify the system to keep track of the number of pages printed since previous refill, and to notify users when queued paper is running low.*)

Fig. 1. A summary description of articulation activities emerging from a resource bottleneck in a task chain.

interruption and render it ineffectual. Then, if office support or maintenance support was available, the user would try to negotiate with the support participant(s) an appropriate enhancement to the text-processing system itself or the work arrangements in the setting, depending on the nature of the bottleneck in the text-processing task chain.

The foregoing scenario highlights the nature of articulation work and its relationship to maintenance work at CSD and CSRO. It illustrates the cooperative activities necessary to deal with unforeseen problems in a person's primary work task chain, i.e., to *re*-articulate the report production task chain. In particular, it underscores the accommodations and negotiations inherent in bringing about commitments to accomplish maintenance activities in a setting. At CSD and CSRO, we found these two basic types of articulation activities:

1) Accommodation activities, whereby participants readily attempt to adapt their patterns of work to the constraints imposed by the current behavior of the text-processing system and other resources involved in the task chain;

2) Negotiation of maintenance activities, whereby participants seek to alter the text-processing system and other resources involved in the task chain to perform according to their expectations or desires.

We next describe each of these two types of articulation activities in turn.[6]

[6]These two types of articulation activities can also be viewed as special kinds of *fitting* work, whereby participants seek to establish a good fit between their patterns of work and the work arrangements in the setting [30], [10].

B. Accommodation Work

Accommodation work emerged when users faced contingencies in their primary work, and responded by doing their work in a way compatible with these contingencies. Researchers at CSD and CSRO considered the production of professional documents an important goal. They sought efficient resources arrangements and effective patterns of work to achieve this goal. Therefore, they eagerly adopted text-processing software tools and resources that could reduce their work. They also preferred to adapt and reshape the way they did their work rather than invest more time, skill, and effort to correct system flaws such as design errors and maintenance inadequacies. Similarly, as shown in the example of the previous section, users accommodated resource contingencies by restructuring their patterns of work.

New CSD and CSRO participants did a good amount of accommodation work when they arrived. They had to adjust their prior patterns of work to fit the local computing arrangements and learn to utilize the available resources as they prepared their reports. Yet there often was no single source of information or documentation which described the use of these systems in a manner adapted to their level of knowledge. Thus they learned to work with these facilities and to (re)structure their work patterns by interactive use (exploration) of the system, or through interaction and negotiation with other computer users.

Both experienced and new users saw their accommodation work as a component of their productivity. Their concern for productivity was reflected in the strategy that guided their accommodation work: they naturally sought to minimize their effort at restructuring their patterns of

work while maximizing the contribution of this restructuring to their primary work activities and purposes.

Accommodation Work at CSD: A common accommodation activity at CSD, which entailed a minimal restructuring of work patterns, was to copy a configuration from an experienced user, especially if it seemed to take care of "just about everything you would want to do for the moment" as one user put it.

Some users went a long way in restructuring their patterns of work. As they became more knowledgeable about the work arrangements, in particular about system limitations and resource constraints, they created their own TEFS configurations. Some participants did so because they believed this would also result in less articulation work thereafter. Others did so to simply avoid the hassle of learning many features that may turn out to be of little use. Still other users engaged in extensive accommodation work in response to compelling institutional constraints, such as the constraint to format their publications according to a particular journal's guidelines. For example, a junior faculty member, Dr. T., brought in files of his unfinished dissertation when he joined CSD. He had to reformat this document with TEFS in order to take advantage of the facility, but needed at the same time to create his own document configuration tailored to the dissertation publication format appropriate to his university of origin.

As newcomers established or restructured their work patterns to take advantage of TEPS, they typically invested as little time as possible in learning useful TEPS features. Instead, they would usually ask an experienced user, but preferably a user who had been at CSD only a short time. This kind of experienced user usually could relate to the new participant's frustrations with more sympathy than an "older" user. More often than not, she/he knew a sufficient yet not overwhelming number of useful features.

Accommodation Work at CSRO: An accommodation activity common to most users was simply to chart out the location of system deficiencies (including "traps," "mazes," and "black holes") as they discovered them and to structure the content of text-processing tasks so as to avoid the charted features.[7]

CSRO participants who were more knowledgeable about system limitations and resource constraints (re)-structured their patterns of work by developing their own combinations of NEAT macro routines. After that, they often believed the additional work of maintaining their individual configuration was worth the savings in articulation work avoided. Some participants also developed their own NEAT configurations to satisfy specific formatting needs by using suitable versions of appropriate programs.

Newcomers at CSRO and other new users of NEAT often preferred to treat NEAT as a black box and rely on the help of experienced users to manage, minimize, or avoid NEAT hassles. As one particpant remarked, "NEAT can be nice. But there are a lot of idiosyncracies that are a pain . . . As I use it more, I run into more of them. If you want to do something exotic, you have to find somebody who knows it. For example, there are some hassles to using special characters. You have to know how to use them. They're not in the manual." This know-how was available only from experienced users.

CSRO participants also engaged in cooperative accommodation activities when dealing with other resources involved in text-processing task chains. For example, unexpected bugs transpired at some point in the behavior of the NETMAIL system. Many of its regular users tried to figure out what had gone wrong with it. Eventually, their cooperation led to a common diagnosis of the problem. However, rather than relying on expectedly lengthy negotiations with the computer facility maintainers and delaying the use of the facility until maintenance work could be performed, users chose instead to continue to use it as it was and accommodated themselves to its erratic, mysterious, and sometimes frustrating performance.

Among the variety of accommodation activities at CSD and CSRO, the most common sought to avoid, circumvent, or undo adverse system effects due to bugs and idiosyncracies in the text-processing system. However, users might eventually believe they simply could not get satisfaction from the system as it was. They would then negotiate system repairs and enhancements. This was a more resource-consuming kind of articulation work, as we show next.

C. Negotiation of Maintenance Work

Use of the text-processing system at CSD and CSRO generated maintenance activities both on the text-processing system itself and on the other computing facilities involved in a document production task chain. We describe in turn the two kinds of maintenance and related negotiations in each setting in turn.

Maintenance Work at CSD: At CSD, maintenance work on the text-processing system itself was performed by one of the researchers, Dr. T. As we saw in looking at participants' accommodation activities, many users started off with TEPS by adopting and adapting another user's configuration as a way of creating their own. These users subsequently maintained these configurations privately and separately from other users. This meant they would spend time and effort on activities that would distract or contribute little to the production of their research publications.

As an illustration, Dr. T. needed to handle the work-in-progress he had brought in from the computer science department from which he was trying to graduate.[8] How-

[7]The development of such accommodation experience and skill is coincidentally a strategy often practiced by users of computer games such as *Adventure*, *Zork*, and other fantasy exploration games.

[8]He would eventually spend more than one year revising his dissertation without completing any other research publications in the interim.

ever, his documents were formatted in macro-routines that did not work on the CSD system, and he did not know how to use either TEDS or TEFS. With the encouragement of two senior colleagues who were enthusiastic users of TEFS on the pay machine, he decided to "bring up" TEFS on the free machine and use it to reformat his documents. He spent much of his time during the Summer of 1984 in reading the manual "from beginning to end." He then realized that he needed to increase the character set supplied by the vendor and that the only way to do so was to augment the TEFS database. He thus had to read the manual for database administrators. Before long, he had learned just about everything there was to learn concerning TEFS. Literally, he had made himself the "resident expert" in TEFS, and had come to be perceived, for all practical purposes, as its *de facto* maintainer. Later, he also came to be considered the expert maintainer of TEDS, the text-editing component of TEPS, through what he saw as his other "contribution to the (research) lab" at CSD. He had developed CEDEX, the integrated mail handling system, by extending TEDS to unify the functions of ZIP and NZIP, respectively, the local mail system and the national network mail system.

As a result of his work, Dr. T.'s expertise in system maintenance was often taken as evidence of official responsibility for enhancement activities. Nevertheless Dr. T. did not consider himself totally bound to maintaining the text-processing system. In the face of too many users' requests for fixes or enhancements, he pinpointed the fact that "most of the work is actually done by a few people." As he put it, "now, I would still give away the knowledge . . . but after all, this is not what I am getting paid to do." He had to balance competing career demands and current work contingencies—i.e., teaching and research (his official primary work) versus maintenance activities. However, he was ready for the right opportunity to engage in a radical enhancement to TEDS, including a complete redesign, which he saw as the prime modification to TEPS really worth making. In the meantime, in negotiating other user requests for a modification, he considered whether there was a general need for the modification, how easily it could be accomplished and, often above all, "how people asked."

Meanwhile, work on the local computing facilities at CSD was the primary work of a resident systems programmer. His primary work included making sure that the facilities were always operational by maintaining the machine, its peripherals, the operating system, and other computing equipment such as personal workstations. In addition, he was officially in charge of maintaining the mail system. He maintained CEDEX (the TEDS-based component of the mail system) in consultation with Dr. T. The systems programmer's decisions on fixes and enhancements were "not based on a vote by the user community" and not very likely to be influenced by negotiating. He would typically make an enhancement on his own initiative, most likely "when it is easy to do, for even the enhancement aspect of maintenance work is often nuisance work." He would then make the enhancement available to the participants who could, at their discretion, either exploit it or do without it. Rarely, he would make an enhancement in response to users' demands, also depending, in his words, on "how people ask and how easy it is to do."

As a rule, neither the resident systems programmer nor Dr. T. would implement a proposed enhancement if it were not intended to compensate for a flaw in the system. Flaws in the system included things such as unintended side effects of a TEDS command or unforeseen consequences of a previous enhancement decision. Both maintainers accorded the lowest priority to flaws which were more readily controllable by the users. The examples of such flaws they cited included patterns of use which did not accommodate the way the system actually worked and proposed enhancements which corresponded to a system "behavior which the user could achieve by some workaround." From the perspective of both maintainers, making enhancements selectively helped prevent the system from going beyond the bounds of manageable complexity.

Eventually, Dr. T. was asked by the senior researchers at CSD to become the main person responsible for the maintenance of the TEPS and all other research support systems at CSD. As he was increasingly spending more of his time and effort in modifying systems that helped him and others produce their research publications, this change of position from faculty to research staff was agreed to after he received assurances from the senior researchers as to CSD's long-term commitment to his new position.

Maintenance Work at CSRO: In contrast to CSD, maintenance activities on the text-processing system at CSRO were distributed across the user community. Maintenance work on NEAT, the dominant text-formatter, was performed solely by the users themselves, with no one in particular having the official responsibility for maintaining the whole system. According to one participant, "actually, it's not maintained at all." Different users indicated that no one at CSRO really knew everything about how the original program worked. Nearly all users reported that NEAT was rife with bugs. Some readily pointed out what they considered to be system design flaws: system bugs, missing features, irrelevant system features, awkward stylistic conventions, and poor system performance characteristics. Some of the system flaws may have resulted from the method of implementation (e.g., insufficient testing) by the system builders. Bugs and idiosyncracies were not fixed, but the system was somehow being used. Most bugs were found through testing with real data. Users attempted an action they believed to be correct, as indicated by someone else or by the system documentation, and found that some unexpected system behavior resulted. The documentation for NEAT was roughly five years old and not updated to reflect maintenance alterations. This led to some problems for users who acquired an intermediate version of the

NEAT program that had been altered without the documentation being changed accordingly.

What resulted from the fact that no one was explicitly assigned to maintain NEAT was that many different sets of NEAT macros were implemented by capable users to get around the bugs they encountered. The resulting routines also had to be maintained. Furthermore, some users were willing to enhance versions of certain programs to satisfy their specific needs. These routines were likely to conform to the particular needs of some, but not all, users. Multiple versions of the NEAT text-formatting system were in use, and their users became system maintainers in order to keep their running version up to date with their patterns of use. In addition, many of the added NEAT macro routines were incompatible or redundant. This gave rise to new system maintenance demands, such as the upkeep of NEAT bug-avoidance macro libraries. Yet, most users felt that the more time and other resources they could avoid spending on support activities, the better. They saw maintenance activities as only a necessary burden, distinguished from "real" computing work, i.e., system use or development.

Maintenance work on the local computing facilities at CSRO was the primary work of a specific support group. The group's official responsibility included maintenance work on the resources supporting test-processing task chains—e.g., machine and operating system, terminals, printers, mail system (NETMAIL), bulletin board (BB), file archiving facility (ARCHIVE), and computer use monitoring system (MONITOR). But the members of this group did not assume responsibility for maintaining NEAT, nor were they involved in acquiring or developing it.

D. Articulation Work Can Become Primary Work

At both CSRO and CSD, we observed that recurring accommodation and negotiation activities could lead to these activities becoming a regular part of someone's primary work. At both CSRO and CSD, a number of users (although not a majority) developed software systems in pursuing the development of their research concepts. For these people, software development work was a form of their primary work, much like their production of research publications. This was no surprise, for in academic computer science organizations such as CSRO and CSD, many of the research publications produced emerged as a result of the development of a software system concept that in turn would be documented in related research publications. However, not all software development work was done to develop research concepts suitable for publication.

At CSRO, a number of junior researchers developed NEAT macro routines to better customize the text processing system to fit both their style of work and the idiosyncracies of this system. In turn, these software routines were usually undocumented, but normally within the immediate comprehension of their user-developers. Accordingly, the circumstances at CSRO were such that developing and maintaining one's own NEAT macro routines became an infrequent, but otherwise regular part of the work researchers would perform in developing their concepts. But no publication we examined mentioned the development of NEAT macros as an essential part of their research concept. Thus, the development of NEAT macros represented a necessary but undocumented part of these researchers' primary work.

At CSD, the circumstances were different. Here we observed that the primary work of Dr. T. grew from his efforts to produce a publishable form of his research results (his dissertation). In order to produce this publication, he had to both modify his source documents and TEPS so that he could format his publication according to another institution's guidelines. He acquired extensive knowledge of TEPS, as well as other electronic mail system components. As it seemed that he was predisposed to modifying these software systems as a major part of his work, then work on these publication support systems was a natural extension to this line of work. When senior researchers recognized Dr. T.'s expertise with TEPS, his disposition towards spending more effort in modifying software systems over the development of research publications, and their need to find someone to maintain the system that supported their primary work, they acted to create a new research staff position that Dr. T. would then occupy. This action reinforced Dr. T.'s preference to develop software over publications, and reduced the senior researchers effort to continually negotiate and accommodate to quirks and features of TEPS. Thus, in this regard, what started as accommodation work for Dr. T. eventually became a regular part of his primary work, via an opportunity to change his job position at CSD.

IV. Discussion

Given the data and analysis for the two cases, we return to our focal interests: understanding the ways local circumstances in the workplace affect how people perform software maintenance work, and the ways maintenance work affects workplace arrangements. We see that local circumstances include the professional incentives that encourage people to recognize opportunities to fulfill occupational or career goals. Maintenance work is performed when bugs are encountered, resource bottlenecks arise, task chains break down, new functionality is desired, related computing innovations are adopted, or personal customizations are sought. The workplace can be described in terms of kinds of problems solved and computing applications used to accomplish primary work, as well as the task chains through which this work is performed. How people perform maintenance work can be characterized according to how they undertake 1) accommodation activities to live with the system as it is, and 2) negotiations with others to alter the system. Last, who performs what maintenance tasks can depend not only on a job title, but also on users' desire to be compatible with other systems to which they are bound through technological, organizational, and professional constraints.

Participants' incentives evolve in response to constraints and opportunities in their computing workplace. The strong emphasis on publishing in academic organizations affects the patterns of use of the text-processing system and the participants' efforts at balancing primary work and articulation work. As we saw at both CSRO and CSD, participants naturally evolve accommodation patterns in response to the day-to-day contingencies which affect their primary work task chains. However, participants sometimes develop patterns of accommodation in a dedicated fashion, such as in the form of a new concept for text-processing system development or modification, in connection with the timely recognition of a pivotal career constraint or opportunity. In contrast to day-to-day accommodation patterns, these latter accommodation patterns can have a substantial impact on the coevolution of the text-processing system, its embedding work arrangements, and its users' patterns of work.

At CSRO, day-to-day accommodation patterns became commonplace, so that maintenance work on the text-processing system by users became an infrequent form of their primary work. Indeed, since members of the CSRO user community were not restricted to using only a single version of the numerous individual subsystems available, all these subsystems had to be maintained. The necessary maintenance activities were thus distributed across the user community throughout the organization.

In contrast, at CSD, the evolution of the text-processing system was intimately tied to accommodation activities that subsequently turned into maintenance activities for one particular user, Dr. T. His text-processing system configuration emerged as *the* TEPS system. Maintenance work on TEPS afforded him a reputation of expertise and eventually became part of his primary work. Dr. T.'s accommodation work on TEFS and his emergence as a *de facto* system maintainer resulted largely from interactions between his career constraints, his perspectives on career opportunities at CSD, and circumstantial computing work arrangements. This is shown by the chronology of his activities:

1) respond to a career constraint—the necessity to reformat his dissertation work-in-progress files with a special TEFS configuration—by investing a substantial amount of time in articulation work to learn TEFS;

2) become knowledgeable enough about TEFS to be considered a *de facto* resident expert;

3) perceive the necessity for an integrated mail handling system supported by TEDS and combining the features of ZIP, the local mail system, and NZIP, the system which separately handles network mail;

4) again, invest time in articulation work and produce CEDEX; in the process become very knowledgeable about TEDS, and be considered, as in the case of TEFS, a foremost authority;

5) balance productive effort between his *de facto* maintenance of TEPS and his primary activities as researcher in ways that often favor the former, fostering a new specialization; and

6) accept the offer of a new position at CSD as research staff, and thereby enable his primary commitment to further develop and maintain CSD's research support systems.

The dynamics of Dr. T.'s commitment to articulation work on TEPS, together with his career constraints and opportunities, created a major shift in the distribution of articulation efforts. His commitment to and routinization of maintenance work on TEPS encouraged other participants to adopt his text-processing system configuration as the common system. As a result, these participants were freed from the burden of doing maintenance work as a notable portion of their articulation work and from dealing with the attendant delays.

Thus, the negotiations and accommodation activities that participants employ to balance their primary and articulation work are mutually bound to how they evolve the systems they use. These activities are also mutually bound to the evolution of work arrangements and the patterns of work.

The participant's proportion of primary work output to articulation work input diminishes when a novel bottleneck in a task chain induces substantial articulation efforts. Eventually, either the bottleneck goes away or the necessary articulation work gets routinized and the participant's proportion of primary work output to articulation work input increases—at least until other kinds of bottlenecks arise in the task chain. However, in connection with the responses of some participants to career contingencies and patterns of system use in the setting, articulation work may also give rise to new primary work along with a centralized organizational unit officially responsible for performing it.

Last, the ratio of primary work to articulation work with a given software system can be viewed as an indicator of how well the system fits into circumstances in the workplace. When primary work dominates, the fit is appropriate, whereas, when articulation work dominates, the fit is poor. Thus, over the lifetime of a system, a high rate of articulation work points to not only a poorly fit system, but also a loss of productive work effort.

V. Related Work

In many ways, our analysis of software maintenance is a departure from established practice. In this regard, our research can be compared to both the approach and results of other studies. For example, the collection of papers appearing in the last two workshops on software maintenance [35], [36] address the applicability and appropriateness of various software tools, techniques, and empirical measures. By and large, these studies focus on *attributes of the software* being maintained, but not on who, how, why and when they are maintained. Nonetheless, these technologies can assist in reducing certain forms of articulation work. For example, tools for maintaining software system configurations and controlling the proliferation of system versions (e.g., [26]) could be employed at CSRO to reduce duplicated maintenance efforts.

However, no technology can completely eliminate or supplant the practical utility of software system articulation work. Instead, each new tool or technique is packaged in such a way that its users must accommodate and negotiate its fit into routine work arrangements.

Belady and Lehman [4], [20] derived through a series of studies a number of laws and dynamics of program evolution from measures of software alterations. Their provocative insights are primarily grounded in attributes of programs and in idealized models of the software development process they conjecture [21]. However, their results give only modest insight into how software maintenance activities vary with the type of organizational setting, the type of application in use, local computing resource availability, incentives and constraints that motivate participants to maintain their systems in idiosyncratic ways, and so forth.

Other studies of software maintenance substantiate the preponderance of the maintenance of existing systems over the development of new systems. Lientz and Swanson [40], [25], [23], [24], [22] document how different kinds of maintenance activities correlate with various organizational attributes and how user requests for system enhancements dominate maintenance activities. But their analyses rely upon data collected primarily from the DP managers' vantage point, and thus provide a restricted view of maintenance. However, in recent reports they recommend that attention be focused on users and their work environment to better understand the dynamics of software maintenance [24], [22].

Rockart and Flannery's [29] studies of end-user computing in large organizational settings find that certain maintenance tasks are frequently performed by functional support personnel in user departments (e.g., Dr. T. at CSD). But their reports do not account for the kinds of computing applications end-users employ to accomplish their routine (primary) work tasks or how these users convince, or otherwise negotiate with, the functional support personnel to get maintenance activities accomplished.

Last, as we found in this and related studies, software maintenance activities add to, or redistribute access to, the available supply of computing resources that systems users can mobilize to accomplish their primary work. Maintenance work of the kinds we describe is central to the ongoing use *and* innovation of local computing arrangements, and it has been observed across many types of organizations and computing applications studied [15], [17], [30], [18], [32]. This leads us to observe that software maintenance work is both a cause and consequence of how systems and work arrangements coevolve.

VI. Conclusions

Software maintenance is a complex and poorly understood phenomenon. Our interest is to better understand how local circumstances in the workplace affect the ways in which software maintenance work is performed. We presented a study of the maintenance of two comparable software systems in two similar organizations. We analyzed the evolution of a software system, its users' primary and articulation work activities, and the work arrangements in which the software system is embedded as three processes that are mutually dependent.

We found there exists a duality between what people want to use their system for versus what they have to do to get their work products out the door. This is what we distinguished as primary work and articulation work. On the one hand, articulation work emerges naturally from bottlenecks in the task chains people follow to accomplish their primary work. On the other hand, the accommodation activities and negotiations employed by system users and maintainers determine the successful completion of their primary work. The pivotal role of articulation work is, not surprisingly, implicit in the traditional distinction [28] between *productive* maintenance activities and "*wheel-spinning*" maintenance activities. The latter's toll increases with the lack of forethought for a software system's maintenance as well as the accommodation efforts imposed by unfamiliarity with the system. But it also increases with the articulation efforts imposed by changing circumstances in the workplace.

Finally, this work poses a number of interesting questions that require further investigation. For example, to what extent are the patterns of software maintenance work described in the cases similar to or distinct from those patterns in other kinds of settings with different software systems? To answer such a question implies a need to examine the primary and articulation work activities of system users, their incentives, computing facilities, and software systems used. In turn, this question further suggests that comparative studies of different kinds of software maintenance work are in order. Also, the kinds of questions we asked—who, what, where, when, why and how—suggest that a simple comprehensive framework for understanding software maintenance work can be developed and serve as a guide for analyzing how well different software tools fit into local circumstances in the workplace.

Appendix

Research Design

The CSRO and CSD case studies were conducted from 1978 to 1984. They are parts of a larger ongoing study of the process of innovation in computing, the routine use of computing systems, and the evolution of software systems in complex organizational settings [17], [30], [18], [31], [34].

Comparative case studies provide a useful way to study poorly understood phenomena [8]. Our research design for developing an understanding of software maintenance follows the methods of case study research. This entails the systematic collection of data about individual cases and the analysis of data to produce generalizable findings [30]. We use empirical data and seek to develop findings based on different levels of analysis to establish a grounded theory [12], [38] of software maintenance.

Our research design incorporates four elements of sci-

entific inquiry: a mode of analysis, terms of analysis, a unit of analysis, and levels of analysis. The *mode of analysis* is the basic framework used to develop theoretical accounts of the phenomena under investigation. Our basic framework is the set of motivating questions presented in the introduction to this paper. The *terms of analysis* reflect the perspective and terminology used for characterizing the phenomena under investigation. Our vocabulary is derived from current research in the social analysis of computing and other forms of technical work [16], [15], [17], [30], [18], [32], [10], [37], [11], [19].

The *unit of analysis* is the subject of investigation used as the basis for theory development. It is the focal element in an individual case study and provides the dimensions of the problem space under investigation suitable for comparative analysis. The dimensions we use are *type of organizations* (academic computer science), *computing application* (automated text-processing), *regular work activities* (producing documents), *computing system in use or contention* (text-processing system), *relevant system life cycle activities* (use and maintenance) and *incentives and constraints on resource use*. In this paper, the focus of our analysis is the work involved in using and maintaining text-processing systems, at both CSRO and CSD.

The fact that both computing applications are text-processing applications and both CSRO and CSD are academic computer science organizations reflects the level of our analysis. The *level of analysis* varies with the dimensions across which the unit of analysis is examined and generalizations of findings are attempted. The unit of analysis (an individual case) stands at the base level of analysis and generalizability. Many choices are possible for the higher levels of comparison and generalizability: cases within the same organization, cases across organizations (our choice here), and cases within and across organizations [30], [8]. In this paper, we examine the maintenance of two computing applications of the same kind (text-processing) in two organizations of the same kind (academic computer science). Further generalization requires, for instance, comparison with similar case studies of different types of applications in different types of organizations [30], [8].

The mode, terms, unit, and levels of analysis chosen provide a framework for the collection and analysis of individual cases selected for comparison and the ongoing synchronization of the comparative analysis. This helps us maintain awareness of where the analysis may lead, how it is pursued, and what is being analyzed, as well as what is being neglected.

In collecting and analyzing data, we focus on the *interactional* nature of work in settings involving computing: how people interact with each other, their computing systems and their work arrangements [17], [30], [18], [10]. In the CSRO and CSD cases, we seek to understand how people in a work setting attempt to resolve unforeseen problems with software system use, such as anomalies in system behavior and other alterations in their computing work arrangements.

The data employed are collected through a sequence of structured interviews with local participants, by first-hand observation of people's activities in the work setting and by participant-observation. The grounded theory approach *emphasizes* collecting information prior to evaluating it, so that no particular theory is singled out to be espoused or discredited [12], [38]. This implies that interviews be started with general questions so as not to preclude any possible outcomes, then continued with more and more focused questions as warranted by the answers obtained.

The interviews we conduct are both structured and open-ended. They are structured by a set of common questions asked of all informants as well as a set of questions specific to each informant. They are open-ended in the sense that these questions are framed so as to elicit descriptive answers. Analysis of these descriptive answers may then lead to further interviews with specific follow-up questions.

At CSRO and CSD, the 50 or so interviews we conducted lasted from as little as 30 minutes to more than 3 hours, with an average of about 50 minutes. Topics covered by the general questions we asked each informant included: 1) how long they had been in the setting; 2) their occupational or professional interests and the nature of their work; 3) which systems they used, for what activities they used them and how they used them; 4) what could go wrong in using these systems to do their work and how they dealt with such contingencies; 5) which changes in the work arrangements had an impact on the way they used the systems to do their work; 6) whether they had any experiences of the ways in which changes actually occurred in the setting; 7) whether and why they would want any changes implemented and how they would go about having them implemented. Naturally, this provides us a wealth of data about many situations which are described in this paper and elsewhere [17], [30], [18], [5], [6].

Informants in each setting were selected because of their role with respect to the systems studied (e.g., users, maintainers, developers), their position in the organizational hierarchy (e.g., managerial staff, faculty, administrative staff), or a referral by another informant.

At CSRO, interviews were conducted with project managers, research faculty, administrative and clerical staff, research associates, research staff programmers, computer facility programmers, the computer facility manager, and graduate assistants. At CSD, interviews were conducted with faculty, administrative, and clerical staff, the systems programmer, and graduate assistants.

Additional data were obtained throughout the study by on-the-spot conversations with participants whenever they happened to encounter problems of system use in the presence of the observer. Moreover, our observation activities included hands-on use of the systems described, review of software system documentation and research publications of all kinds, administrative memoranda, and bulletin-board and other public electronic mail messages.

ACKNOWLEDGMENT

Helpful comments on an earlier draft were generously provided by D. Estrin, L. Gasser, E. Gerson, S. L. Star and A. Strauss. The referees also suggested important clarifications in our presentation.

REFERENCES

[1] R. S. Arnold and D. A. Parker, "The dimensions of healthy maintenance," in *Proc. 6th Int. Conf. Software Eng.*, pp. 10–27, Sept. 1982.

[2] R. Balzer, T. E. Cheatham, and C. Green, "Software technology in the 1990's: Using a new paradigm," *Computer* (*Special Issue on The DoD STARS Program*), vol. 16, no. 11, pp. 39–45, Nov. 1983.

[3] V. R. Basili and A. J. Turner, "Iterative enhancement: A practical technique for software development," *IEEE Trans. Software Eng.*, vol. SE-1, Dec. 1975.

[4] L. A. Belady and M. M. Lehman, "A model of large program development," *IBM Syst. J.*, vol. 15, no. 3, pp. 225–252, 1976.

[5] S. Bendifallah, "Management of computing: A case study," in *Case Studies in the Management of Computing*, W. Scacchi, Ed., Dep. Comput. Sci., Univ. Southern California, Los Angeles, Tech. Rep., 1983.

[6] S. Bendifallah and W. Scacchi, "Software evolution and articulation work: A comparative case study," in *Proc. Int. Workshop Development and Use of Computer-Based Systems and Tools*, Aarhus, Denmark, Aug. 1985, pp. 59–82.

[7] B. Boehm, "Software engineering," *IEEE Trans. Comput.*, vol. C-25, pp. 1226–1241, Dec. 1976.

[8] F. van der Bosh, J. R. Ellis, P. Freeman, L. Johnson, C. L. McClure, D. Robinson, W. Scacchi, B. Scheff, A. von Staa, and L. L. Tripp, "Evaluation of software development life cycle: Methodology implementation," *ACM SIGSOFT Software Eng. Notes*, vol. 7, no. 1, pp. 45–60, Jan. 1982.

[9] J. H. Fujimura, "The construction of doable problems in cancer research," *Social Studies of Sci.*, to be published.

[10] L. Gasser, "The social dynamics of routine computer use in complex organizations," Ph.D. dissertation, Dep. Inform. Comput. Sci., Univ. California, Irvine, 1984.

[11] E. M. Gerson and S. L. Star, "Analyzing due process in the workplace," *ACM Trans. Office Inform Syst.*, vol. 4, no. 3, pp. 257–270, 1986.

[12] B. Glaser and A. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago, IL: Aldine, 1967.

[13] R. L. Glass and R. A. Noiseux, *Software Maintenance Guidebook*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[14] T. Kidder, *The Soul of a New Machine*. New York: Avon, 1982.

[15] R. Kling, "Social analyses of computing: Theoretical perspectives in recent empirical research," *ACM Comput. Surveys*, vol. 12, no. 1, pp. 61–103, Mar. 1980.

[16] R. Kling and E. M. Gerson, "Patterns of segmentation and intersection in the computing world," *Symbolic Interaction*, vol. 1, no. 2, pp. 24–43, 1978.

[17] R. Kling and W. Scacchi, "Computing as social action: The social dynamics of computing in complex organizations," *Advances in Computers*, vol. 19, pp. 249–327, 1980.

[18] ——, "The web of computing: Computer technology as social organization," *Advances in Computers*, vol. 21, pp. 1–90, 1982.

[19] B. Latour and S. Woolgar, *Laboratory Life*. Beverly Hills, CA: Sage, 1979.

[20] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proc. IEEE*, vol. 68, pp. 1060–1076, Sept. 1980.

[21] ——, "Program evolution," *Inform. Processing Management* (*Special Issue on Empirical Foundations of Information and Software Science*), vol. 20, no. 1-2, pp. 19–36, 1984.

[22] B. P. Lientz, "Issues in software maintenance," *ACM Comput. Surveys*, vol. 15, no. 3, pp. 271–278, Sept. 1983.

[23] B. P. Lientz and E. B. Swanson, *Software Maintenance Management*. Reading, MA: Addison-Wesley, 1980.

[24] ——, "Problems in application software maintenance," *Commun. ACM*, vol. 24, no. 11, pp. 763–769, 1981.

[25] B. P. Lientz, E. B. Swanson, and G. E. Tompkins, "Characteristics of application software maintenance," *Commun. ACM*, vol. 21, no. 6, pp. 466–471, June 1978.

[26] K. Narayanaswamy and W. Scacchi, "An environment for the development and use of large software systems," in *Proc. Softfair II*, San Francisco, CA, Dec. 1985, pp. 14–25.

[27] G. Parikh, "The world of software maintenance," in *Techniques of Program and System Maintenance*, G. Parikh, Ed. Cambridge, MA: Winthrop, 1982, pp. 9–13.

[28] R. S. Pressman, *Software Engineering—A Practitioner's Approach*. New York: McGraw-Hill, 1982.

[29] J. F. Rockart and L. S. Flannery, "The management of end user computing," *Commun. ACM*, vol. 26, no. 10, pp. 775–784, Oct. 1983.

[30] W. Scacchi, "The process of innovation in computing: A study of the social dynamics of computing," Ph.D. dissertation, Dep. Inform. Comput. Sci., Univ. California, Irvine, 1981.

[31] W. Scacchi, Ed., "Case studies in the management of computing," Dep. Comput. Sci., Univ. Southern California, Los Angeles, 1983.

[32] ——, "Managing software engineering projects: A social analysis," *IEEE Trans. Software Eng.*, vol. SE-10, pp. 49–59, Jan. 1984.

[33] ——, "A software engineering environment for the system factory project," in *Proc. 19th Hawaii Int. Conf. Syst. Sci., Software*, vol. IIA, 1986, pp. 822–831.

[34] W. Scacchi, S. Bendifallah, P. Garg, A. Jazzar, J. Macias, *et al.*, "Modeling the software process: A knowledge-based approach," Dep. Comput. Sci., Univ. Southern California, Los Angeles, 1986.

[35] R. S. Arnold, Ed., *Rec. Software Maintenance Workshop*. Washington, DC: IEEE Computer Society Press, 1983.

[36] *Proc. Conf. Software Maintenance*. Washington, DC: IEEE Computer Society Press, 1985.

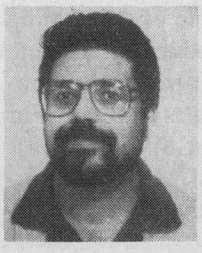
[37] A. Strauss, "Work and the division of labor," *Sociological Quart.*, vol. 26, no. 1, pp. 1–19, 1985.

[38] A. Strauss, *Qualitative Analysis*. New York: Cambridge University Press, 1987.

[39] A. Strauss, S. Fagerhaugh, B. Suzcek, and C. Weiner. *The Social Organization of Medical Work*. Chicago, IL: University of Chicago Press, 1985.

[40] E. B. Swanson, "The dimensions of maintenance," in *Proc. 2nd Int. Conf. Software Eng.*, 1976, pp. 492–497.

[41] W. K. Wiener-Ehrlich, J. R. Hamrick, and V. F. Rupolo, "Modeling software behavior in terms of a formal life cycle curve: Implications for software maintenance," *IEEE Trans. Software Eng.*, vol. SE-10, pp. 376–383, July 1984.



Salah Bendifallah received the Engineer diploma in engineering economy/informatics from the Polytechnic School of Algiers, University of Algiers, Algeria, in 1974; the M.S. degree in industrial engineering from Stanford University, Stanford, CA, in 1979; and the M.S. degree in engineering/computer methodology from the University of California, Los Angeles, in 1982.

He is completing work toward the Ph.D. degree in computer science at the University of Southern California, Los Angeles. His major research interests are in knowledge-based modeling and simulation of the software process, knowledge-based software engineering, and social analysis of software engineering work.

Mr. Bendifallah is a student member of the American Association for Artificial Intelligence, the Association for Computing Machinery, the Society for Computer Simulation, and the Society for General Systems Research.

Walt Scacchi (S'77–M'80) received the B.A. degree in mathematics, the B.S. degree in computer science in 1974, and the Ph.D. degree in computer science from the University of California at Irvine.

He is an Assistant Professor of Computer Science and Communications at the University of Southern California, Los Angeles. Since 1981, he has directed the System Factory Project at USC. His research interests include large-scale software engineering, knowledge-based systems, and social and organizational analysis of computing.

Dr. Scacchi is a member of the Association for Computing Machinery, the American Association for Artificial Intelligence, and the Society for the History of Technology (SHOT).