The Evolving Object of Software Development

By

Paul S. Adler

Volume 12(3): 401–435
ISSN 1350–5084

organization

articles

The Evolving Object of Software Development

Paul S. Adler

University of Southern California, USA

Abstract. *This paper contributes to an ongoing debate on the effects of bureaucratic rationalization on relatively non-routine, knowledge-work activities. It focuses on the Software Engineering Institute's Capability Maturity Model (CMM®) for software development. In particular, it explores how the CMM affects the object of software developers' work and thereby affects organization structure. Empirical evidence is drawn from interviews in four units of a large software consulting firm. First, using contingency theory, I address the* technical *dimensions of the development object. Here CMM implementation reduced task uncertainty and helped master task complexity and interdependence. Second, using institutional theory, I broaden the focus to include the* symbolic *dimensions of the object. Adherence to the CMM involved the sampled organizations in efforts to ensure certification, and these symbolic conformance tasks interacted in both disruptive and productive ways with technical improvement tasks. Finally, using cultural-historical activity theory, I deepen the focus to include the* social-structural *dimensions of the object. Through these lenses, the software development task appears as basically contradictory, aiming simultaneously at use value, in the form of great code, and at exchange value, in the form of high fees and profits: the CMM deepened rather than resolved this contradiction. The form of organization associated with these mutations of the object of work is a form of bureaucracy that is simultaneously mock, coercive, and enabling.* ***Key words.*** *bureaucracy; cultural-historical activity theory; interdependence; Marxism; object; software development*

DOI: 10.1177/1350508405051277 http://org.sagepub.com

Schumpeter (1942/1976) famously argued that large firms would learn to routinize innovation. A small stream of organizational research has explored how firms have successfully introduced discipline into the unruly process of innovation (e.g. Griffin and Hauser, 1992; Craig, 1995; Adler, 1999; Wheelwright and Clark, 1992; Jelinek and Schoonhoven, 1993; Bart, 1999; Barley and Orr, 1997; Cardinal, 2001; Organ and Green, 1981; Podsakoff et al., 1986; Alvesson and Kärreman, 2004; Clark and Fujimoto, 1991; Nixon, 1998; Davila, 2000). The larger body of organization theory, however, has been deeply skeptical of Schumpeter's prediction, arguing that rationalization is incompatible with the creativity required for innovation (Burns and Stalker, 1961; Mintzberg, 1979; Hall, 2001; Merchant, 1998; Ouchi, 1979; Raelin, 1985). To this contingency-theoretic skepticism, recent work in institutional theory adds that firms adopting formalized, standardized approaches to innovation tasks are often merely seeking symbolic legitimacy and seek to preserve their innovative capability by buffering their technical core from such stifling bureaucracy (Scott, 1995, 2003; Meyer and Rowan, 1977; Meyer et al., 1983; Westphal et al., 1997).

The software arena provides a rich context for exploring this debate. As software systems have grown larger and more complex, so too has the proportion of projects that fail to meet their goals or fail entirely (Gibbs, 1994). The Standish Group's characterization of the 'chaos' in software bears repeating, notwithstanding the methodology weaknesses (Standish Group, 1994). Data were collected on 8330 industry software projects in 365 firms in banking, manufacturing, retail, wholesale, health care, insurance, and government. The projects were relatively large: the average project cost ranged from US$0.4 million in small companies (US$100–200 million revenue) to US$2.3 million in larger companies (over US$500 million in revenue). Overall:

- only 16 percent of projects were on time and within budget and met originally specified requirements—only 9 percent in large companies;
- 31 percent of projects were 'impaired' and eventually canceled;
- the remaining 53 percent of projects were 'challenged,' and the average challenged project met only 61 percent of its requirements;
- combining the impaired and challenged categories, the average such project was 189 percent over budget and 222 percent over schedule.

Today we face what many observers describe as a veritable crisis in software development (Lieberman and Fry, 2001; Jones, 2002). One popular approach to taming this chaos is a focus on 'process,' understood as the bureaucratic standardization, formalization, and management control of work processes (other approaches will be discussed below). And one increasingly popular vehicle for attaining greater process discipline is the Capability Maturity Model (CMM®) developed by the Software Engineering Institute (also described in more detail below). At high levels of 'maturity,' software development should, its proponents argue,

resemble a factory process in the discipline of its operations and the predictability of its results.

This bureaucratic rationalization, however, encounters considerable skepticism. Skepticism has focused on two main issues, echoing the concerns of the large body of organization theory referred to above. First, there is skepticism about the CMM's motivational effects. In the current state of technology, the software process cannot be entirely automated. As a result, software developers' commitment and creativity are still needed in order to master the non-routine aspects of the process. Second, there is skepticism about CMM's conformance effects. Some observers argue that, when organizations focus on conformance to externally defined process standards such as the CMM, they lose focus on real improvements to their processes and products. Typical of the opposition to standardized and formalized methodologies is this assessment by two well-respected software management experts:

> Of course, if your people aren't smart enough to think their way through their work, the work will fail. No Methodology will help. Worse still, Methodologies can do grievous damage to efforts in which people are fully competent. They do this by trying to force the work into a fixed mold that guarantees a morass of paperwork, a paucity of methods, an absence of responsibility, and a general loss of motivation. (DeMarco and Lister, 1987: 116)

We currently lack empirical research on the effects of such rationalization on developers' work activity or on the corresponding changes in the structure of development organizations. The present paper aims to help fill these gaps through an analysis of four units in a large software consulting firm that has adopted the CMM. The goal of the paper is simultaneously to characterize changes in software development organizations and help advance organization theory.

My premise is that, to understand how management approaches such as the CMM affect organizations, we need to understand these approaches' effects on the object of work itself. My theoretical starting point is the concept of the object as developed by cultural-historical activity theory (CHAT). (On CHAT and closely related variants of practice theory, see Engeström, 1987, 1990b; Cole, 1996; Leont'ev, 1978; Engeström et al., 1999; Chaiklin et al., 1999; Wertsch, 1979; Blackler, 1993; Holt and Morris, 1993; Nardi, 1996). As the dictionary and CHAT both tell us, an 'object' is simultaneously (a) something given to the mind or senses and (b) a purpose. The object of the blacksmith's activity is simultaneously a piece of iron, an inert mass, and the mental image of the shape it should take, a goal. Indeed, it is the tension between the two that motivates the blacksmith's activity and thus serves as a starting point for understanding the form of organization assumed by that activity (Engeström, 1990a; on CHAT's view of the object, also see Foot, 2002).

In the sections that follow, I first discuss this CHAT conceptualization of the object and how it relates to other organization-theoretic approaches. I then characterize in general terms the object of software development activity, provide some background on the CMM, and describe my research methods and context. The body of the paper reports on how the CMM has changed the object of software development in the sampled organizations and the corresponding changes in the structure of development organizations.

The Concept of the Object

CHAT takes inspiration from Marx in both its epistemology ('dialectical materialism') and ontology ('historical materialism'). Epistemologically, CHAT begins with the first of Marx's Theses on Feuerbach:

> The main defect of all hitherto-existing materialism . . . is that the Object [*der Gegenstand*], actuality, sensuousness, are conceived only in the form of the object [*Objekt*], or of contemplation [*Anschauung*], but not as human sensuous activity, practice [*Praxis*]. (Marx, 1845/2002)

Marx here is criticizing both simplistic materialism—where the object is merely a given in the external world (German: *das Objekt*)—and idealism—where the object is only ever our mental construction of it, the subjective meaning and purpose we attribute to it (German: *die Anschauung*). Arguing a thesis close to that of the American pragmatists, Marx proposes a dialectical synthesis of these two philosophical traditions, postulating that the most fruitful starting point for an understanding of the world is our practical engagement with it. The object of our sensuous, practical activity (*Praxis*) is always simultaneously an independently existing, recalcitrant, material reality *and* a goal or purpose or idea that we have in mind. This mix is nicely captured in the German term *der Gegenstand*: *gegen* means against, towards, contrary to, signaling a reality that offers resistance to our efforts and desires, and *der Stand* means category or state of affairs.

The history of organization theory recapitulates a similar dialectic of epistemologies. In a first phase of organization research, contingency theory focused on how the effectiveness of given organization forms depended on the nature of the organization's 'task,' where task was an exogenous given in the form of the key demand placed on the organization by both the material world and higher-level strategic choices. CHAT takes on board contingency theory's core intuition that the task/object plays a key role in determining the effectiveness of different organizational forms.

However, contingency theory focused on only the *technical* dimensions of tasks; in effect, it adopted a simplistically materialist view, focusing on the *Objekt* as a given material reality. In a second phase, the field of organization theory was enriched by a range of approaches that broadened the focus to address the political, cultural, symbolic, and

socially constructed nature of the task. These perspectives have come together most recently under the umbrella of institutionalization theory, which teaches us that the organization's understanding of its tasks is often in practice taken for granted, and that, even when this understanding is conscious, the task is often guided as much by concern for symbolic legitimation as by technical efficiency (see surveys by Scott, 2003; Powell and DiMaggio, 1991). CHAT takes on board too this recognition of the social construction of the task and the problematic of the object/task's taken-for-grantedness and contestability.

However, although institutional theory captures well the object as culturally constructed *Anshauung*, it loses sight of the recalcitrant material reality of the *Objekt*. Moreover, institutional theory has been largely silent on how legitimacy concerns come to shape action—silent, that is, on the subject's practical engagement with these symbolic challenges (for an exception, see Brunsson et al., 2000). Following Marx, CHAT's view of the object preserves both its objective and its subjective dimensions. Contingency theory's simplistic materialism and institutional theory's cultural idealism both capture facets of the object as it presents itself in practical activity. The activity system is marked by the tensions between these two facets.

CHAT also takes inspiration from Marx's ontology, challenging organization theory to go deeper and to theorize the *social-structural* constitution of the object/task. CHAT argues that the ontologies of both contingency theory and institutional theory are too flat and therefore fail to grasp the deeper, intrinsically contradictory structuring of the object of work activity in capitalist firms. Marxist social theory is layered rather than flat, in the sense articulated by Bhaskar (1993): the empirically observed world is the result of the overdetermination of multiple layers of structure. To say, as institutionalization theory does, that organizations face both technical and symbolic legitimation challenges is to provide a richer empirical characterization than contingency theory, but does not provide much more insight into the underlying forces shaping those challenges.

Here CHAT takes up Marx's analysis of the capitalist production process. CHAT reminds us that the object of work in a capitalist firm is a 'commodity,'[1] and as such embodies two, contradictory, goals: the creation of use value and the creation of exchange value. According to Marx, the production process within capitalist firms has two aspects (reflecting the two aspects of the commodity): the *labor process*, in which use values in the form of work effort, tools, and materials combine to create new use values, and the *valorization process*, in which these use values appear in the form of exchange values, and in which the operative considerations are not technical but monetary—wages, capital, and profit (Marx, 1977: appendix; Thompson, 1989; Bottomore, 1983: 267–70). The objects of capitalist work are typically therefore deeply contradictory.[2] Where institutionalization theory sees conflicts between the technical and symbolic

aspects of the object, CHAT sees use-value/exchange-value contradictions inherent in both aspects, and it is this underlying contradiction that more profoundly shapes the direction of, and provides the force for, change in work and organization.

The Object of Software Development: Generic Features

These three theoretical perspectives on the object each offer useful vantage points for the study of software development activity. Let us begin with contingency theory.

In its discussion of task, classic contingency theory bequeaths us a rich vocabulary for characterizing the object of work activity.[3] Starting with Woodward, Lawrence and Lorsch, Perrow, and the Aston team, and subsequent work by Van de Ven and numerous others, several dimensions of task have been found to be salient in understanding how the work tasks of an organization shape its structure. I focus on the three dimensions that are perhaps the most commonly cited: uncertainty, complexity, and interdependence (see Scott, 2003: ch. 9; Zigura and Buckland, 1998). Uncertainty (closely related to novelty, analyzability, and ambiguity) is the difficulty of solving the problems posed or of resolving the exceptions encountered in the work. Complexity is the number of different types of problems posed in the work. Interdependence is the extent to which the solutions to these problems are interrelated.[4]

Software development—in particular the development of large-scale systems that has proven so chaotic—is notable for simultaneously high uncertainty, high interdependence, and high complexity. The standard, contingency-theoretic recommendation for managing such tasks is to increase the skill (professionalism) of the personnel and to assure these personnel considerable autonomy in their work through the adoption of an organic organization design.

The software crisis shows that this recommendation has failed. Software already relies on highly trained personnel, and the empirical studies show that more or less organic forms of organization are the norm in this industry. The problem is that the organic organization form does not scale well. As software products and projects have grown in scale, development organizations have been forced to look for other solutions. The range of potential solutions can be presented in terms of whether they address primarily complexity, interdependence, or uncertainty. In addressing these three dimensions, organizations can seek either to *reduce* the challenge and/or to improve their capacity to *manage* the challenge. Let us take the three dimensions in turn, and we will see that beaucratization has become a technical necessity.

Organizations can try to reduce the challenge of *complexity* through *automation*. Higher-level languages, code generators, and sophisticated programming environments can take over the task of managing much of the complexity of software development. Bureaucratic rationalization is a

way to manage complexity once it exceeds the cognitive limits of the individual expert; automation will reduce the need for this bureaucracy. However, there are limits to our ability to automate when uncertainty is high, as is typically the case in software development (Weber, 1997).

A second approach is to reduce the challenge of *interdependence* through *modularity* (e.g. Baldwin and Clark, 2000). The burden of managing interdependence can be alleviated by careful pre-specification of interface standards. This enables the overall system to preserve high levels of complexity and allows individual modules to incorporate high levels of creativity. This approach is currently attracting considerable attention as the premise underlying the open source software movement. Microsoft's 'feature-driven design' approach relies on a similar logic (Cusumano and Shelby, 1995). However, as J. D. Thompson (1967) argued, only modest degrees of interdependence can be managed by pre-specified standards. Moreover, pre-specifying interfaces limits more radical and architectural innovation. More intensive forms of interdependence require mutual adjustment and, when combined with complexity, require detailed planning. Mutual adjustment suffices only on a small scale with modest levels of complexity—conditions that do not obtain in large-scale software projects.[5] Planning takes us to bureaucracy, to which we now turn.

The third approach focuses on reducing *uncertainty* through *bureaucratic rationalization*. Here the focus is on standardizing and formalizing not only module interfaces but the entire work process. The goal is to maintain or to increase the creativity of the output but to reduce the uncertainty of the tasks leading to that output. Once uncertainty is reduced, then a combination of automation and bureaucracy can be used to manage greater levels of complexity and interdependence. Bureaucracy is far more popular than modularity in the world of large-scale custom software systems development, because the individual client is typically unwilling to pay for an upfront interface standardization effort whose benefits will flow to many other clients too. In this world, the largest part of the software industry, developers therefore have no choice but to manage interdependence and complexity rather than try to reduce them, and this means in turn that they have no choice but to find ways to use bureaucracy to reduce uncertainty. These conditions have helped motivate the growing popularity of the CMM.

It is not only the object of software development that has evolved; our theory of the object has evolved too. As indicated in the previous section, recent developments in organization theory suggest two major extensions to the classic contingency-theoretic analytic apparatus. First, institutional theory has alerted us to the importance of symbolic legitimacy in shaping organization structure. The task of the organization is not merely technical—assuring effective transformation of inputs into outputs—but also symbolic—ensuring its legitimacy as an institution in the eyes of both external and internal stakeholders. When software development

organizations adopt the CMM, the object of work expands to include ensuring conformance to CMM standards.

There is considerable debate over how technical and symbolic tasks are related. In a strong version of institutional theory, task and object are always understood as socially constructed and culturally defined—as *Anschauung*. A weaker version sees technical and symbolic tasks as coexisting facets of reality, even if over time, within a given institutional field, the symbolic tends to displace the technical as the main driver of organizational change. As will become clear in the following section describing the CMM, the pressure for symbolic conformance is of considerably importance in some sectors of the software industry, and my research therefore sought to understand how the CMM's symbolic effects interacted with its technical effects.

CHAT offers a further extension to organization theory's view of task, first by integrating the objective and subjective dimensions, and second by characterizing the specific social form in which the tasks of work activity present themselves in capitalist firms. The fundamental contradiction characteristic of capitalism between use value and exchange value is expressed in the tensions created by conflicting goals of high-quality software code and high profits in the exchange of this code for fees. The 'substance' of use value (in this case, code, documents, files, etc.) serves as the substratum for the 'social form' of exchange value (capital, wages, profit), but the two layers—substance and form, use value and exchange value—obey heterogeneous, conflicting imperatives.[6] My research therefore also aimed to understand how this contradiction shaped the adoption, implementation, and effects of the CMM.

To characterize the impact of bureaucratic rationalization on software development, I shall review changes to the object of development work from each of these three perspectives in turn—technical, symbolic, and social-structural—then characterize the resulting form of organization. First, however, I describe in more detail the CMM and the context of my research.

Bureaucratizing Software Development: The CMM

In the 1980s, the US Air Force studied 17 major software systems contracts and found that every one was late (by an average of 75 percent) and over budget (Humphrey, 2002). In 1984, frustrated with such chaos, the Department of Defense (DoD) funded the Software Engineering Institute (SEI), based at Carnegie-Mellon University, to develop a model of a more reliable software development process. With the assistance of the MITRE Corporation, SEI developed a 'capability maturity model' (CMM), releasing a preliminary description in 1987 and the first official version (version 1.1) in 1991 (see Paulk et al., 1993a,b). The software CMM was subsequently complemented by CMM tools for systems engineering, people management, and software acquisition. In 2000, several of these were integrated into a broader tool called CMM-Integration.

This study focuses on the software CMM. This CMM distinguishes five successively more 'mature' levels of process capability, each characterized by mastery of a number of Key Process Areas (KPAs)—see Table 1. The CMM belongs to a class of improvement approaches that focus on 'process' rather than 'people.' It does not recommend any particular approach to organizational and behavioral issues: it focuses on the 'whats' and not the 'hows,' leaving CMM users to determine their own implementation approach. Level 1 represents an ad hoc approach: an organic organization design would rate as a Level 1. Level 2 represents the rationalization of the management of individual projects. At Level 3, standard processes are defined and used for the organization's entire portfolio of projects. Level 4 pushes rationalization even further, specifying mechanisms for quantifying the development process. Level 5 specifies systems for assuring the continuous improvement of that process. The underlying philosophy of this hierarchy was inspired by Crosby's (1979) five stages of TQM maturity (see Humphrey, 2002; a bibliography on the CMM is available at http://www.sei.cmu.edu/docs/biblio.pdf). In its recommendations for the standardization and formalization of work and management practices, for the specialization of line and staff roles, and for hierarchical structuring, the CMM represents a strong program of bureaucratic rationalization.

Early CMM assessments revealed a startlingly 'immature' state of software process: 80.0 percent of the 132 organizations assessed during 1987–91 were found to be at the 'ad hoc' Level 1, 12.3 percent at Level 2, and only 6.9 percent at Level 3, 0.0 percent at Level 4, and 0.8 percent at Level 5. Such immaturity, we should note, is entirely consistent with contingency theory's recommendation of the organic form. Over subsequent years, however, there appears to have been significant shift towards higher maturity levels (although it is difficult to tell, given the changing and unrepresentative nature of the sample, which is composed of organizations that volunteer for evaluation). Of the 1343 organizations assessed between 1999 and 2003, 13.3 percent were at Level 1, 43.5 percent at Level 2, 25.6 percent at Level 3, 8.5 percent at Level 4, and 9.2 percent at Level 5 (Software Engineering Institute, 2004). This shift was assisted by the fact that the DoD and other government and private sector organizations began using Software Capability Evaluations (SCEs) based on the CMM as part of their source selection process. The first evaluations pressed suppliers to reach Level 2, but before long the bar was raised to Level 3. Not surprisingly, the CMM has become the basis for numerous software service organizations' improvement efforts in both the government and commercial sectors. (The CMM is almost unknown among firms developing prepackaged software products; see Software Engineering Institute, 2004.)

Evidence is slowly accumulating that moving up the CMM hierarchy leads to improvements in product cost, quality, and timeliness.[7] On the other hand, and echoing the broader literature, there is considerable

Table 1. The CMM Model

Level	Focus and description	Key Process Area
1: Initial	Competent people and heroics: The software process is ad hoc, occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.	
2: Repeatable	Project management processes: Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.	• software configuration management • software quality assurance • software subcontract management • software project tracking and oversight • software project planning • requirements management
3: Defined	Engineering processes and organizational support: The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.	• peer reviews • intergroup coordination • software product engineering • integrated software management • training program • organization process definition • organization process focus
4: Managed	Product and process quality: Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.	• software quality management • quantitative process management
5: Optimizing	Continuous process improvement: Improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.	• process change management • technology change management • defect prevention

concern over its bureaucratic nature. (In this ambiguity, CMM resembles the broader family of 'software factory' concepts of which it is a part; on the concept of software factory and the associated debates, see Cusumano, 1991; Swanson et al., 1991; Griss, 1993; Weber, 1997; Friedman and Cornford, 1989; Greenbaum, 1979; Kraft, 1997.) In partic-

ular, concerns are often expressed that the discipline recommended by the CMM will reduce the autonomy of developers and will therefore be experienced by them as burdensome and coercive constraint. This would stifle the motivation and creativity that are, over the longer run, required for high-quality and innovative software development (e.g. Crocca, 1992; Bach, 1994, 1995; Conradi and Fuggetta, 2002; Lynn, 1991; Ngwenyama and Nielson, 2003). Numerous commentators have also expressed concern that the symbolic conformance pressure will displace attention to real technical improvements and that it imposes on firms a single vision of the improvement process (e.g. Bollinger and McGowan, 1991; Saiedian and Kozura, 1995). One software development manager interviewed in the present study expressed these concerns this way:

> Programming has always been seen as more of an art form than a factory process. Programmers are supposed to be creative, free spirits, able to figure things out themselves. So the software factory idea was very alien to the culture of programmers. (A: department manager)

Research Context and Methods

To explore CMM-driven bureaucratization of software development, I studied one of the larger US-based professional service information technology firms, which I will call GCC. With the support of senior management, interviews were conducted with staff in four of its Programs involved in government work. (A 'Program' is an organizational unit devoted to ongoing work with a single client.) Two Programs, A and C, were at CMM Level 5 and two sister Programs, B and D, were at Level 3. An appendix to this paper provides background on these four Programs.

Like the bulk of the software industry, these Programs were developing systems that embodied considerable real innovation but also involved a considerable mass of rather more routine work. The proportion of routine activity was driven primarily by the high volume of software maintenance activity. The proportion of non-routine activity was driven primarily by customer demands for ever-higher levels of system performance and an accelerating flow of new programming languages and technologies. On balance, developers' tasks embodied enough uncertainty to make employee motivation a continuing priority, and this in turn gave the employees real power in their relations with management:

> Buy-in is important in this kind of business. Take an example: programming languages. The DoD was very enthusiastic a few years ago about Ada. It was a great language from a management point of view, since it specified things in a way that gave management a lot of control over the process. But the programmers preferred C because it was less constraining and more open. They simply refused to get on board with Ada, and management lost the fight. (A: Program manager)

Coercion was a costly policy in dealing with highly skilled professional staff on whose initiative and goodwill the organization's success depends:

> In this organization, you have to lead by example. A military command style just doesn't work. We've had a few managers with that more military approach, and we've had to ease them out of their roles. We've reorganized to give more responsibility to the better managers. The ethos here is that people do what it takes. So you have to lead, not command. A while back, we were running late on an important project. The manager tried to impose a 50 hour week schedule. It really hurt morale—people were already working 60 hours a week! (D: development manager)

Nevertheless, GCC, like other larger organizations, deployed a whole hierarchy of increasingly fine-grained standard operating policies to create process discipline:

1. 'policies' defined universal requirements;
2. 'processes' defined the methodologies that structured all the projects;
3. 'procedures' defined activities within the project; and
4. 'instructions' defined requirements at the individual task level.

The 'granularity' of process discipline at its finest levels can be gauged by the instructions at Program C. Separate instructions covered high-level design, two types of low-level design, two types of code reviews, testing, change request implementation and resolution, and root cause analysis. Each instruction was several pages in length. They typically included the specific forms to be completed as well as flow-charts detailing the sequence of steps involved. Overall, the written processes summed to some eight linear-feet of shelf space. In recent years, a growing proportion of this documentation appeared in the form of on-line databases, and more of the procedures specifying work-flows were being built into automated collaboration systems.

Process maturity meant a great volume of documentation not only to read but also to write, and to write in a highly standardized manner. In the words of one interviewee, perhaps exaggerating for effect:

> I can write the code in two hours, but then I have to spend two days documenting it. It can be very frustrating. We have to document check-in and check-out, a detailed design plan, a development plan. We have to print out all the differences between the old and the new code. There's documentation for inspection and certification. There's an internal software delivery form. A test plan. And all these need to be signed . . . I used to be an independent developer for about three years. I never even created a flowchart of my work! The only documentation I needed was a 'to do' list. So I had to change of lot of habits when I got here. (B: developer)

The CMM encouraged the GCC Programs to refine and extend this bureaucratic apparatus. Program A illustrates the effectiveness of this push. Program A had worked with the CMM to improve its software process since 1991. They had made only modest changes in process

technology, but had nevertheless cut development costs for comparably complex projects by 60 percent over 10 years. Over the period 1994–8 alone, the cost of software development for the average project was reduced by 23 percent, from 255 staff-months to 200, and productivity increased from 475 lines of code per staff-month to 560. Error rates in the late 1970s were about 7 errors per 1000 lines of developed code; by the end of the 1990s the average rate was 0.7. Project cycle times were reduced by between 5 percent and 20 percent over the most recent five-year period. And the accuracy of planning was considerably increased: the average effort variance (=(actual–estimated)/estimated) fell from 40–50 percent in the early 1990s to less than 20 percent in the late 1990s.

Contingency theory (Burns and Stalker, 1961; Lawrence and Lorsch, 1967; Galbraith, 1977) teaches us that the scope for process standardization and formalization such as recommended by the CMM is closely related to the degree of routineness versus the uncertainty of the organization's key tasks. In some GCC Programs, the basic tasks were more routine than in other Programs; indeed, the level of detail in the process varied across Programs according to the average routineness of their tasks. In comparison with Program C, for example, its sister Program D dealt with a broader range of technologies and these technologies evolved more rapidly. Not surprisingly, Program C was considerably more mature in its process than Program D, and its process was more controlled at finer degrees of granularity.

Within Programs too, the tasks of different departments differed in their degree of uncertainty. For example, at Program D, one department was responsible for defining site requirements, planning and procuring hardware, getting it to the site, and installing it, and this department, unlike the development department, did have detailed, formalized, standard operating procedures. At the other end of the spectrum, the engineering and technology (E&T) department at Program A handled tasks that were relatively much less routine than the Program's other departments. The E&T department manager explained:

> Because of the nature of our work perhaps, we don't have anywhere nearly as much process as Development. We don't usually have a fixed delivery date. [. . .] Yes, we have to manage to our budget, but our process is only a few pages long. (A: project manager)

Notwithstanding these differences, all four Programs had pursued CMM certification and, as a result, all the departments in all four Programs had achieved high (Level 3) or very, very high (Level 5) levels of bureaucratization. Moreover, they had done this while keeping personnel turnover below the industry average, filling open positions relatively quickly, and maintaining relatively modest levels of compensation and limited financial incentives (as compared with software organizations that produced software products or provided services to commercial sector clients).

In order to understand how these efforts had transformed the development process, I interviewed between 15 and 22 people at all levels in each of these four Programs. Interviews lasted approximately one hour. They were tape-recorded and interviewees were assured anonymity. The recordings were transcribed and edited versions were sent back to interviewees for review and correction. I also consulted voluminous internal documentation from each of these Programs, as well as documents from corporate entities supporting them.

The Influence of the CMM on the Object of Software Development

The Technical Dimension: Reducing Task Uncertainty

If regularities can be found in an innovative task, the same degree of novelty in the task's *output* can be achieved with less uncertainty in the task *process*, increasing the latter's efficiency and stability. Interviewees argued that it was possible to impose more discipline on the more routine parts of the work without impairing, and indeed improving, the flexibility needed for the less routine parts:

> Even when tasks are more innovative, you can still get a lot of advantage from process. You need to sit down and list the features you want in the system, and then work out which are similar and which are different from your previous work. And you need to make sure the differences are real ones. You'll discover that, even in very innovative projects, most of the tasks are ones you've done many times before. Then, for the tasks that are truly novel, you can still leverage your prior experience by identifying somewhat related tasks and defining appropriate guidelines based on those similarities. They won't be precise instructions of the kind you'll have for the truly repetitive work: but these guidelines can be a very useful way to bring your experience to bear on the creative parts of the work. (B: testing, formerly with Program A)

The standardized, formalized process fostered by the CMM stabilized the object of work, which meant less chaos and more intelligibility:

> Before I came to GCC, I worked for a very small software firm doing business software. It was what the SEI folks would call a Level 1 organization—completely ad hoc. No documentation, no design reviews, no standardized testing. So there was a lot of chaos and rework. . . . In a place like this, everything is more organized, and you know exactly where you are in the development process. I like the fact that you know where you're up to and how to do your work. It's more streamlined and there's less rework. (C: developer)

One important effect was improvement in the organization's capacity for planning, which helped reduce trial and error in resource allocations and schedules:

> Our process is so mature that we have great estimating data. We can develop a really accurate estimate for a bid on a project in a matter of hours. (A: developer)

Mastering Complexity and Interdependence

Reducing uncertainty facilitated the management of the interdependence and complexity that are involved in large-scale systems development. The CMM also contributed more directly to this mastery by 'expanding' the object socially, both in its mental representations and in its objective reality.[8] Instead of each functional department working on its own task in siloed isolation, the tasks were now part of an integrated whole:

> Our policies and procedures mean that I have better information on what we're trying to do because we have better requirements documents and better information on how to do it with Instructions etc. At Level 5 versus Level 1, I'm more confident we're all playing to the same sheet of music. Looking across the organization, process also means that managers understand better the way the whole system works, so they are all playing the same game. . . . That gives me more confidence that my piece will fit with the rest of the system. (C: developer)

This social expansion of the object was directly related to the technical expansion represented by the greater documentation burden in the developers' work. Many interviewees saw this documentation as the necessary support for an imaginary dialogue with previous and future developers and with other people who are working on the code:

> I think that our process—and even the paperwork part of it—is basically a good thing. My documentation is going to help the next person working on this code, either for testing or maintenance. And vice versa when I'm on the receiving end. (C: developer)

Process maturity also expanded the object of work to include dialogue with customers on requirements and progress monitoring. This facilitated coordination between organizations:

> One of the main reasons I came back from the client to Program A was because I like working in a more mature organization. When I worked at the client, it was a real eye-opener—and very frustrating. We'd been working for four years, and reached a major milestone, and only then discovered that the system didn't work. The contractor [that we were working with] had managed to hide the problems from the customer [i.e. us] for four years! Their process was so weak that they couldn't manage the project on their end nor provide us with good progress data. The chaos was terrible, and it showed up in atrocious [personnel] turnover—over 40 percent per year! Once I got back to Program A, the contrast was really striking. We knew exactly where we were at any given time, what we were doing, and why. (A: testing)

The Symbolic Dimension: Assuring Legitimacy

The commitment of these GCC Programs to the CMM meant in practice that the object of work also expanded to include ensuring conformance to the CMM standards and ensuring successful certification of that conformance. Process conformance therefore competed with technical performance priorities:

> Lower-level managers juggle the needs of the customer and the pressures from GCC upper management. And upper management is focused primarily on things that strengthen GCC's position for obtaining future work rather than what we need to retain current work. So, even though the requests for things like CMM ratings may have no value-added for our immediate assignment, we do them anyway. (A: developer)

These competing priorities reflect the tension between technical and symbolic demands as described by institutionalization theory. Clearly, part of the CMM effort was 'for show,' at least in its earlier phases and in the less mature organizations:

> The evaluation and CMM SCE forced us to update our documents. We didn't really change anything in how we work though. (D: developer)

> Before the evaluation in May and the Action Team work got underway, I was pretty skeptical about our Program management's commitment to moving further ahead on process. I just didn't think management would take it seriously. I was sure it was a one-time effort to get past the evaluation and then we'd drop it. That's happened before: we run around to document stuff because we have to get past some evaluation, then it's dropped. The bottom line here is the product. That's how we get paid. Process and support activities are basically just overhead. So I was pleasantly surprised to see that senior management really was committing to this as a new way of doing business. (D: project support staff)

The challenge of CMM certification was to 'map' the Program's existing practices to the CMM's Key Process Areas (KPAs, as described in Table 1). In some cases, this mapping exercise revealed existing practices to be satisfactory, and the mapping was therefore experienced as a wasteful burden; in most cases, however, the CMM provided guidance for process improvement efforts. Program C had long worked under Department of Defense Military Standards for quality, so the discipline of the CMM was experienced against that backdrop, but its experience was otherwise representative:

> Most of our CMM work has been focused on translating what we already do into the CMM KPAs. We were doing virtually all the KPAs anyway, just because you can't manage large-scale projects without doing something like what the SEI is recommending. The first SCE team told us they knew that we must have good procedures and that everyone followed them because everyone told us the same thing; but, they said, the process must have been tattooed on the inside of eyelids because they couldn't find them written down anywhere. So we spent the next year putting them down on paper. For example, we had an informal training and mentoring program and, when we got serious about the CMM, we wrote it down. Writing the process down has had some great benefits. It's made us think about how we work, and that's led to improvements. For example, formalizing the training program has helped bring some outliers into conformance. And we formalized the SEPG [Software Engineering Process Group] process, and that has helped stimulate improvement. (C: training staff)

As predicted by institutionalization theory, organizations under conformance pressure sought to buffer their core by creating specialized staff roles. Program A used process engineering and quality assurance (QA) staff as 'shepherds' to help line groups prepare for external process audits:

> The shepherds were the greatest gift. We could rely on the shepherd to explain to us the CMM KPAs and work out with us how we could satisfy them. And we could vent all our frustrations on them too! Each time, the evaluation was a really irritating interruption. But there was a concerted effort to keep them out of the way of the technical people. (A: developer)

Symbolic versus Technical Tasks Institutionalization theory often takes a cynical view of the role of symbolic conformance; among interviewees in these four organizations, assessments were more nuanced. Many felt that the CMM offered a compelling model of a better process (what Engeström calls a 'more advanced model' of the activity system):

> The CMM is helping us move ahead. Just to take an example, even if the CMM didn't exist we would need a technology change management process [a Level 5 KPA]. Of our 450 people, we have about 50 people in CM [configuration management], QA, and data management. To move them from one process to another is sometimes like herding cats! The CMM helps us in that by providing an industry-validated approach. (C: Program manager)

In their debates against proponents of alternative organizational development scenarios, proponents of the CMM had the advantage of this cultural-historical validation.[9] Other interviewees expressed more negative views of the conformance task. One concern was that the CMM prescribed certain features of the development process and, in doing so, substituted its own 'wisdom' for the results that might emerge from a more self-directed organizational learning process.[10] This excerpt illustrates:

> SEI has encouraged people to think that progress will come from 'implementing' the KPAs, when you really need to decide which KPAs matter to your business and how you should pursue them. Many organizations, even some people in our Government Systems Group, think they need to implement all the requirements of every Level. So the CMM ends up being seen as externally coercive rather than as an internally motivated improvement process. You can get a false sense of security when you force your way to certification—or a false sense of failure if you try to force your way and fail. (B: process engineer, formerly with Program A)

A second concern—expressed by some of the interviewees in the two Level 5 Programs, A and C, albeit only a minority—focused on the value of some Level 4 and 5 practices:

> We struggled to get past Level 3. Level 3 seems to give you most of the CMM's benefits. Frankly, Levels 4 and 5 haven't changed or helped much. Beyond 3, documenting the technology management process didn't really

> do much for us: we manage to change technology pretty effectively without formalizing that process. But on the other hand, defect prevention has been very useful. (A: contract officer)
>
> I think Level 3 was worth doing. But most of Levels 4 and 5 just don't seem to add much. It isn't about everyday stuff anymore. We are doing most of these processes, and documenting them adds a lot of cost but not much value. (C: quality assurance staff)

Notwithstanding these reservations, the upside of external legitimacy pressure was that this pressure could facilitate internal change:

> I can see that external evaluations are a very important learning tool. It's just like in college: 90 percent of what the student learns is in the week before the test! So we do need the test to create that incentive. But it's not an end in itself. The real issue is: Is passing the test just a veneer? That depends on how the managers treat the test—as an opportunity to put some banners on the walls, or as an opportunity to focus attention and get some real learning done. At Program A, we have reached (well, almost reached) the point where people like the tests as an opportunity to show off their improvements. (B: process engineer, formerly with Program A)

The Social-Structural Dimension: Software's Use Value vs. Exchange Value

CHAT reminds us that the object of developers' activity at GCC was not only to produce use value—code and CMM certifications—but also, and more fundamentally, to produce exchange value—to generate fees and profits. This section reviews in turn how the CMM helped support the creation of use value and how the use-value/exchange-value contradiction shaped the object of work.

Creating Use Value On their first exposure to a high level of maturity, many software developers felt dismayed that they now had to conform to alien rules and procedures where they had previously enjoyed great autonomy and had simply relied on their professional skills to create high-quality code. But, after gaining experience with the more bureaucratic process, many of my interviewees had come around to seeing this formalized process as a better way to create better code. Interviewees often expressed themselves in terms such as these:

> Where I used to work before I came to GCC, the development process was entirely up to me and my manager. What I did, when I did it, what it was going to look like when it was done, and so forth, was all up to me. It was very informal. Here everything is very different. It's much more rigid. It's much more formal. A lot of people lay out the schedule, the entire functionality, and what I'm going to be accountable for—before I even get involved. . . .
>
> When I got here I was kind of shocked. Right off, it was 'Here are your Instructions.' 'So what does this tell me?' 'It tells you how to do your job.' I thought I was bringing the know-how I'd need to do my job. But sure enough, you open up the Instructions, and they tell you how to do your job: how to lay the code out, where on the form to write a change request number, and so on. I was shocked.

> But I can see the need now. Now I'm just one of 30 or 40 other people who may need to work on this code, so we need a change request number that everyone can use to identify it. It certainly feels restrictive at first. They explained the Instructions and the whole Program C process to us in our orientation seminar, but it's hard to see the value of it until you've been around a while. Now I can see that it makes things much easier in the long run.
>
> I hate to say it. As a developer, I'm pretty allergic to all this paperwork. It's so time-consuming. But it does help. You've got to keep in mind, too, that, by the time we see the Instructions, they've been through a lot of revision and refinement. So they're pretty much on target. (C: developer)

Reactions such as these need to be put in historical perspective. Some years prior to my research, rationalization had taken a form many developers experienced as alienating and coercive. Program C was an early adopter of process discipline in its earlier, pre-CMM incarnations in the Department of Defense Military Standards, and these standards afforded little opportunity for developer participation: '[Military Standard] 2167A was supposed to make coding a no-brainer' (D: development manager). But by the time of my study a decade or more later, the Level 5 Programs had pushed the rationalization of the process further, and the process had taken a more participative and enabling form:

> The first phase, in the late 1980s, was conformance. We had developed our standard process—a big fat set of requirements and standards—and most managers felt that it was just a matter of ensuring that people were implementing it. The second phase, in the early 1990s, was enlightenment. This phase coincided with our big TQM push. We started getting working-level people involved in improving things. The third phase, running between about 1994 and 1998, was empowerment. The word might sound trite to some people, but we had the process framework, and we had the involvement, so we were really ready to delegate more autonomy down to the projects and the tasks. (A: Program manager)

Participation was a key factor in ensuring that developers saw the formalized process as supporting rather than merely constraining their work. Process maturity—as these GCC units had implemented it—had expanded the object to include not only inter-functional coordination and documentation but also process development:

> As compared to say 20 years ago, I think there's more involvement by developers in defining and tailoring project S&Ps [standards and procedures]. Back then, we pretty much accepted them passively, like rules of the road when you're driving—even when we had very definite opinions about them! (A: systems engineering)

The most advanced form of this combination of discipline and participation was the 'tailoring' process, formalized in the Tailoring Cycle. Here, the project manager was supposed to consult with the project staff on how the established development standards and procedures might need to be modified to accommodate the specific requirements of the project:

> We always say at Program A that 'People support what they help create.' That's why the Tailoring Cycle is so important. As a project manager, you're too far away from the technical work to define the S&Ps yourself, so you have to involve the experts. You don't need everyone involved, but you do need your key people. It's only by involving them that you can be confident you have good S&Ps that have credibility in the eyes of their peers. (A: project manager)

The Tailoring Cycle was not the only vehicle for participation in process definition. In Programs C and D, management encouraged staff to send suggestions to the Software Engineering Process Group (a standing committee of senior line and staff managers). Moreover, in all four Programs, but particularly in the two Level 5 ones, developers were regularly involved in process improvement projects. In Program C, for example, a recent survey revealed that 19.5 percent of the personnel had been involved in at least one such effort in the course of the previous year. The CMM Level 5 KPAs required a formalized process improvement process, and GCC implemented this requirement in a highly participative manner.

Exchange Value versus Use Value

The contradictory nature of the commodity object was visible in several tensions. First, and most generally, interviewees were often aware that their process improvement efforts were all at risk of being overridden by a higher imperative. As the interviewee quoted earlier put it:

> The bottom line here is the product. That's how we get paid. Process and support activities are basically just overhead. (D: project support)

The object of work was therefore fractured along hierarchical lines:

> As I see it, GCC is a corporation, and that means it's run for the benefit of the major stockholders. So top management is incentivized to maximize dollar profits. Quality is only a means to that end and, in practice, quality sometimes gets compromised. I used to be a technical person, so I know about quality. But now I'm a manager, and I'm under pressure to get the product out—come what may. I just don't have time to worry about the quality of the product. I have a manager of software development under me who's supposed to worry about that. (D: development manager)

These tensions were particularly visible to the interviewees in the form of missed opportunities for process improvement:

> One key challenge is maintaining buy-in [for process improvement work] at the top. Our top corporate management is under constant pressure from the stock market. The market is constantly looking at margins, but Government business has slim margins. That doesn't leave much room for expenditures associated with process improvement—especially when these take two or three years to show any payoff. (C: process engineering manager)
>
> We could do better at capturing and using lessons learned. We have all the vehicles for doing it—presentations, newsletters, databases. But it takes

> time. And there are so many competing priorities. In the end, it's all about profit and meeting schedules [laughs]! (A: project manager)

This contradiction was also visible in the gap between the expanded object of work as an enriched use value and the limited investment in tools and staff resources as costly expenditures that reduced exchange value:

> There's no doubt that more process maturity means more paperwork. Some of it is good, and some of it is an impediment, especially from a productivity point of view. Unless we have the tools to automate this documentation, it has to slow us down. We still don't have the right tools. (C: project manager)

> The key issue moving forward, I think, is that we still don't have the resources we need to devote to process. A Program of this size should have a full-time staff dedicated to our internal process maintenance. (C: quality assurance staff)

In sum, the CMM deepened rather than resolved the contradiction between use value and exchange value. The contradiction was certainly not resolved: the CMM expanded the object, but the constraints of 'profits and meeting schedules' remained in place, constantly engendering new tensions. Indeed, the contradiction had deepened: the constraints of profitability were increasingly—and more obviously—harmful to quality and process improvement efforts.

This latter proposition requires elucidation. In the earlier history of the software industry, it was the imperatives of 'profits and meeting schedules' that drove the managerial innovations needed to master larger, more complex, software projects. The constraints facing these innovations were those created by recalcitrant programmers defending their autonomy. Greenbaum quotes a programmer in the 1970s:

> We never really established this as a policy, but what kind of job security would we have if we wrote everything down the way they wanted us to? We didn't like it when things got too out of control, but on the other hand would you see to it that your job was so standardized that it could be done by a monkey? (Greenbaum, 1979: 75)

As we saw, '[Military Standard] 2167A was supposed to make coding a no-brainer.' Nevertheless, in this early period, the exigencies of exchange value were progressive forces, breaking down obsolete craft models of organization that were incapable of mastering the challenges of larger-scale, more complex systems development. Exchange value and use value expanded together.

In the later period, by contrast, process discipline was redefined, refined, and internalized by the 'collective worker' as a way to master their expanded collective task. Limits on organizational learning shifted from those created by individualistic developers defending their autonomy to those created by the fundamental structure of the capitalist enterprise and its subordination to the profit imperative. Developers had,

over this period, become increasingly conscious that the key issue was not how to preserve their individual autonomy (and how to reconcile this defense with their frustration when 'things got too out of control'), but rather a deeper one— how to abolish the fetters created by the capitalist form of enterprise.

Bureaucracy: Simultaneously Mock, Enabling, and Coercive

The previous three sections have sketched the mutations of the software development object under the influence of the CMM; in this section, we ask: to what form of organization does such an object correspond?

Clearly, these were highly bureaucratized organizations, in the technical sense of the term. The CMM had led to an increase in standardization, formalization, specialization, hierarchical differentiation, and the creation of specialized staffs. But what *kind* of bureaucracy was this? In his classic study, Gouldner (1954) identified three forms of bureaucracy. In a 'mock' bureaucracy, rules are promulgated for their symbolic value but ignored in practice (e.g. no-smoking rules). In a 'representative' bureaucracy, rules serve the interests of both managers and workers and are typically developed jointly (e.g. safety rules). In a 'punishment-centered' bureaucracy, rules serve to legitimate management's right to sanction workers in areas of labor–management conflict (e.g. taking company property for personal use). These distinctions are echoed in more recent research on the symbolic aspects of bureaucratization highlighted by institutionalization theory and on the distinction between enabling and coercive bureaucracy developed in some of my own work. The analysis presented in the preceding sections suggests that we should not interpret these three 'patterns' as mutually exclusive; rather, they coexist. This coexistence, however, demands theorization.

Mock and *Real*

The bureaucracy of these GCC units was simultaneously mock and real. The units were simultaneously concerned with the symbolic effects and the technical effects (either punishment or representative) of their formalized development process. The possibility that symbolic and technical pressures can operate simultaneously is acknowledged by Scott (2003); but in his account, as in many others in the institutionalization theory perspective, the symbolic aspects tend to displace the technical. In particular, much work in institutionalization theory has propagated the idea that late adopters of organizational innovations are motivated primarily by symbolic conformance benefits and use various buffering techniques to avoid any change in their technical core. Whereas GCC's Program A was a relatively early adopter, its other Programs were not; nevertheless, in all four cases we saw genuine efforts to improve the technical core, efforts that were both stimulated and dampened by institutional conformance pressure. The cynical pathos of much institutionalization theory is belied by the fact that sometimes the technical and

symbolic functions reinforced each other, as when an upcoming assessment prompted technical improvements.

Coercive and *Enabling*

The coexistence of a punishment-centered and a representative function presents a deeper conceptual challenge. This coexistence is rarely explicitly acknowledged in prior research. My own earlier work (Adler and Borys, 1996; Adler, 1999), for example, presented enabling and coercive forms as alternatives on a continuum. On the evidence of the previous section, such a representation misses something more fundamental: both functions of bureaucracy coexist and, moreover, this coexistence is of a very different nature than the coexistence of mock and real forms of bureaucracy.

The CHAT perspective invites us to see the representative/enabling and punishment/coercive functions of bureaucracy not merely as two considerations that are potentially conflicting but potentially reinforcing—as is the case with technical versus symbolic functions—but rather as two sides of a real contradiction, reflecting the fundamentally contradictory character of the object of production as use value and exchange value. On the one hand, the production process is a collective labor process, and its organization under the bureaucratic model represents an effort to facilitate that collaborative endeavor. On the other hand, the production process is a valorization process in which capital invested in equipment, materials, and wages seeks a profit, and its bureaucratic organization represents a system for coercing surplus labor from recalcitrant workers. The tensions that derive from this contradiction are not amenable to quantitative trade-offs: they are fundamentally disruptive since the two poles are qualitatively incommensurable.

The contradictory coexistence of enabling and coercive aspects of bureaucracy was visible in several use-value/exchange-value tensions in the GCC Programs I studied.[11]

Differentiation vs. Integration On the one hand (the use-value aspect), and consistent with the social expansion of the object, process maturity meant closer coordination with others in a broader range of functions. Although great emphasis was put on standards and plans as ways to ensure this coordination in a cost-effective manner, mutual adaptation requiring direct communication had also been strengthened (referring to Thompson's, 1967, three generic coordination mechanisms):

> Process means that people play more specialized, defined roles, but also that these specialists get involved earlier and longer as contributors to other people's tasks. If we analyzed the way a coder uses their time, and compared it with comparable data from, say, 15 years ago, we'd find the coder doing less coding because of more automated tools. They'd be spending more time documenting their code, both as it was being built and afterwards in users' guides. They'd be spending more time in peer reviews. And they'd be spending more time in design meetings and test plan

> meetings. As for testers, we've consolidated some phases of testing. We used to do integration test, system test, operations test and acceptance test. Now we just do integration test and then a final independent test. That has reduced redundancy and the cost of testing, but now the testers are more involved in system concept definition and requirement definition activities. (A: quality assurance staff)

On the other hand (the exchange-value aspect), this more differentiated and more integrated division of labor was pregnant with contradiction, because, in a capitalist firm, competitive rivalry between departments creates enormous centrifugal forces.[12] Indeed, cross-department coordination was a common problem for these GCC Programs, as is typically the case in larger, more complex capitalist firms. This is illustrated by the problems in the link between systems engineering (which was responsible for concept identification, system architecture specification, requirements specification) and software engineering (software design, code, unit test and integration). The situation at Program C was typical:

> On most of our projects, different people fill the two roles, systems engineering versus software engineering. (On smaller projects, the same person may have both roles.) As with any interaction between two groups, there have been communication gaps between them. There are a variety of reasons: the systems engineers point to the software engineers and say 'They didn't read what I wrote,' and 'They don't understand what I mean,' and the software engineers point back and say 'They didn't specify the requirements adequately,' 'The requirements are inconsistent,' and 'That wasn't in the requirements.' It gets even more challenging when the requirements changes keep coming up to the day before delivery. (D: process engineer, also works with Program C)

Community vs. Autocracy On the one hand (the use-value aspect), GCC managers understood that process maturity required a high level of employee motivation and participation; on the other hand (the exchange-value aspect), the vertical authority structure characteristic of the capitalist firm—reflecting the fact that managers are agents not of employees but of owners—created a constant risk that managers would veer off from collaboration into coercion. GCC management was constantly struggling to avoid this costly eventuality, but it was a struggle they could never entirely win:

> By and large, we haven't had too much difficulty bringing our managers around to this more collaborative approach. But we choose our project managers with an eye to their commitment to collaboration too. We did have a problem with one staff person. He had a very difficult relationship with the project people he was supposed to be helping. We got a lot of complaints that he was trying to force the projects to conform to his idea of how they should function. We tried to counsel him and get him to work in a more cooperative way. But he just wouldn't ease up. Eventually we just had to let him go. And we had quite a battle with one Program manager when he wasn't picked to head a new project: we felt he just wasn't enough of a team manager. We didn't initially have any questions on the employee

> survey about your boss. Frankly, people were worried that managers might retaliate. But now we do, and we find the data very useful in surfacing management problems. The earlier rounds of the survey did show some big communications problems in some groups. Counseling often helped, and in some cases we moved people out to other positions. (A: Program manager)

This contradiction was also visible in tensions that disrupted collaboration with clients. The CMM encouraged extensive coordination and collaboration with the client (the use-value aspect), but divergent performance pressures (the exchange-value aspect) could easily pull the parties apart:

> The biggest problem here has been the customer and getting their buy-in. At Program A, our customer grew towards process maturity with us. Here [at Program B], we started with a less mature client. Some of the customer management even told us that they didn't want to hear about QA or our quality management system—they saw it as wasteful overhead. When you bid a project, you specify a budget for QA and so forth, but if they don't want to pay, you have a resource problem. And once you get the contract, then you start dealing with specific project managers within the customer organization, and these managers don't necessarily all believe in QA or simply don't want to pay for it out of their part of the budget. On the Y2K project, the customer kept changing standards and deadlines. Basically, we were dealing with a pretty process-immature customer, and that made it difficult for us to build our process maturity. Things have improved considerably since then. (B: process engineer, formerly with Program A)

Enabling vs. Coercive Rules On the one hand (the use-value aspect), the fabric of rules that coordinated work in these bureaucracies supported an enabling function; on the other hand (the exchange-value aspect), these very same rules supported coercion:

> I think formalized process and metrics can give autocratic managers a club. But it also gives subordinates training and understanding, so it makes the organization less dependent on that manager: he can be replaced more easily. Before I came to GCC, I worked for one of the most autocratic managers you can find. It was always, 'And I want that report on my desk by 5 p.m. today,' with no explanation or rationale. Compared to that kind of situation, an organization with a more mature process leaves a lot less room for a manager to arbitrarily dictate how you should work and when work is due. And a more mature process also means that there are more formal review points, so any arbitrary autocratic behavior by a manager will become visible pretty quickly. (D: quality assurance staff)

Conclusion

This study of the CMM's impact on the object of software development yields a complex image. The technical dimensions of the object were transformed so as to reduce task uncertainty and to facilitate the mastery of complexity and interdependence. The symbolic dimensions of the object acquired greater importance owing to the pressure to achieve

formal certification, and this both hindered and helped efforts to improve on the technical dimensions. The social-structural dimensions of the object continued to express the fundamental contradictions of commodity production, and the CMM served to deepen rather than resolve those contradictions.

With organization theory's progressive eclipse of contingency theory, we have tended to lose sight of the object of work—particularly its technical, objective, obdurate objectivity—as a factor in organization structuring. Cultural-historical activity theory is a way of retrieving the crucial insight offered by contingency theory, while preserving the insights of institutional theory, thus both broadening and deepening our understanding of the nature of the object and its effects on organization.

Appendix: Background on the Four Programs Studied

Program A: CMM Level 5

Program A has had a continuous contractual relationship with its customer for 30 years. Many employees have been attracted to Program A because of the high public profile of the customer. Historically, Program A has about 20 programs under way at any one time, each building and maintaining mid-sized (100–400,000 lines of source code) subsystems for the complex infrastructure required by the customer. Program A has relied mainly on established technology; however, over recent years, the Program's tasks have become more complex as the customer requirements and the associated technologies have evolved. The business environment has also become more demanding, with considerable pressure for more code reuse and tighter deadlines.

Unlike the other three Programs, direct customer pressure was not the proximate cause of Program A's commitment to the CMM. The Program's management saw the adoption of the CMM as an opportunity to improve their development process and as a way to add credibility for potential future customers. In 1991, the first formal, external Software Capability Evaluation (SCE) rated the organization at Level 1. The organization subsequently undertook several internal self-assessments. In 1996, the second evaluation rated it close to Level 3. In 1998, it was assessed as a Level 5 organization.

Over the past decade of process improvement efforts, Program A has seen its average effort variance reduced by over half. Average error rates have been reduced by 75 percent (and 50 percent in the past five years). Productivity has shown a consistent 6 percent annual rate of improvement.

Program B: CMM Level 3

Program B's mission is to build information resource management tools for its government client to use in operations around the world: internal accounting, management information support, and so on. Program B's

staff develop new systems, maintain and upgrade old ones, and operate a Help Desk.

GCC won the contract in 1998 by promising to leverage GCC's experience in Program A to help Program B reach CMM Level 3 within 18 months. GCC replaced nearly 30 contractor organizations that worked largely independently of each other. Program B itself employs directly or indirectly about 300 people. The largest of its sub-programs employs about 90 people building and maintaining a system of some 700 files, comprising about 1 million source lines of code (MSLOC).

To help reach Level 3, several people were transferred from Program A. The two largest programs were each led by former Program A people, and Program A veterans staffed several other key management and staff positions. These people used Program A's process as a starting point, and Program B managers were mobilized to tailor this process to their requirements.

Program B's process efforts were slowed down by a very difficult Y2K program, which strained relations with the client. That completed, relations improved and the Program was officially assessed as Level 3 in early 2000.

Program C: CMM Level 5

This Program, like Program A, has had a continuous contractual relationship with its DoD end-customer for some 30 years. But the relationship had always been mediated by other organizations serving as prime contractors.

Program C employs some 400 technical staff. It undertakes two to five major programs at time, each representing about 2.5–3.2 MSLOC. These programs create new versions of the weapons control systems they provide to the DoD. Internally, Program C is divided into four main units that develop and maintain the main modules of the system, plus several support departments.

The key drivers of process maturity at Program C have been the succession of Military Standards imposed by the end-customer (the government) in conjunction with the intermittent pressure of their immediate customer (the prime contractor). By the middle of 1998, Program C was evaluated at Level 4, with all but some minor elements of Level 5 in place as well. In 2001, it was evaluated at Level 5, and has maintained this rating in subsequent evaluations. The quality of its products was widely recognized: the program customer satisfaction index consistently averaged over 97 percent.

Program D: CMM Level 3

Program D developed infrastructure systems for the DoD. GCC developed its proposal in 1987 and the contract opened in 1991. It had developed 2 million lines of operational code over the 1993–9 period. Program D is unusual within GCC because it covers the whole product lifecycle,

offering complete solutions including hardware, the integration of hardware and software, warehousing, installation, and ongoing support. Program D is also unusual within GCC for its extensive use of commercial off-the-shelf hardware and software. Its systems incorporate over 200 commercial products, and are being used in about 100 sites, of which about 50 are interlinked. In 2001, Program D employed some 350 people directly, plus a further 120 contractors.

Traditionally, software process has received less attention in this Program than in the other three we studied. However, as part of a bid for a very large DoD contract, Program D had to undergo an external process evaluation. In preparation for that evaluation, it conducted its own assessment, and discovered that the Program would likely be rated no higher than Level 1. As a result, the general manager chartered an Improvement Team and charged it with taking the Program to Level 3. QA was significantly strengthened—the staff grew from three to eight people—and a broad effort at process documentation was undertaken throughout the organization by department-level Action Teams. By the end of the 1999, the Program was assessed as Level 3.

Notes

1 Following classic political economy, Marx uses the term 'commodity' not to imply standardization as distinct from customization (as suggested by current colloquial use) but rather to refer to any object produced for sale rather than for direct use.

2 CHAT follows Marx, who himself followed Hegel, in taking contradictions to be a feature of objective reality rather than purely notional in the mind of the observer. Contradiction here is a relation between two real forces, not merely a logical relation between two propositions. As such, contradictions are the source of change (see Redding, 2002; Ilyenkov, 1982).

3 There is considerably terminological confusion surrounding all this in the literature. First, there is confusion between task and technology. CHAT's roots in psychology invite us to differentiate task (object) as the work to be done from technology (tools) as the means of doing it, and from worker (subject) as the agent endowed with certain capabilities. Conventional organization theory is less discriminating in its micro foundations and more interested in the determinants of organization structure, and thus folds task, tools, and worker skills together under the broader heading of 'technology' (see e.g. Scott, 2003). Adding to the confusion, some authors refer to this aggregate not as technology but as task. The term 'environment' is also often used to denote something close to task. The logic here is that, in the structural-functionalism that marks much organization theory, the primary 'task' of the organization and of the worker is to adapt to the demands imposed by the environment.

4 There is some confusion in the literature because uncertainty is often used to designate the total effect of all three of the task dimensions and sometimes just the first of these. We should note that the literature on the task environment highlights three related dimensions: munificence, dynamism, and complexity (see Harris, 2004). Munificence of the external environment

corresponds to slack in the organization; as such, it is an important variable affecting organization structure but one that speaks to the context of work rather than to the work itself. Dynamism matters because of the problem-solving difficulty it creates, and thus corresponds to the uncertainty dimension. We should also note yet another set of dimensions commonly used where organization theory has been affected by economic agency theory. Here the focus is on the dimensions of task that are particularly relevant to the design of control and incentive systems, namely behavior observability (or meterability) and outcome observability. When combined with variables such as the controller's knowledge of the transformation process and outcome importance, these dimensions can be shown to correlate with organization structure (e.g. Kirsch, 1996). However, observability is a rather malleable factor; so research along these lines typically finds strong correlations but affords little insight into causality.

5 'Agile programming' approaches such as extreme programming and feature-driven development seem to offer the promise of process improvement without the perils of bureaucracy. In an increasingly turbulent competitive world, these perils weigh heavy on the minds of many software managers. But, as Boehm and Turner (2003) argue, agile methods have proven appropriate only where systems and development teams are small, where the customers and users are available for frequent consultation, and where requirements and the environment are particularly volatile.

6 The CHAT view highlights the use-value/exchange-value contradiction underlying both the technical and symbolic aspects of the object discerned by institutionalization theory. Institutionalization research often highlights the tension between technical tasks (which are typically seen only in their use-value aspect, as if only productive efficiency mattered in this technical domain) and symbolic tasks (which are typically seen only in their exchange-value aspect, as if only external control mattered in this symbolic domain). CHAT brings to the fore the two parts missing from that picture: the exchange-value aspect of technical tasks, where coercive control is key, and the use-value aspect of symbolic conformance, where learning for efficiency is the driving factor.

7 The Software Engineering Institute website lists several case studies of high-maturity organizations and the benefits they have achieved. (Case studies of Level 4 and 5 organizations are presented in *Crosstalk*, 1999; Dutta and Van Wassenhove, 1998; Humphrey et al., 1991; Diaz and King, 2002; Diaz and Sligo, 1997; Pitterman, 2000; Keeni, 2000; Wigle and Yamamura, 1999; McGarry and Decker, 2002; Butler and Lipke, 2000; Willis et al., 1998; see also the reports on the SEI 'High-maturity workshops' conducted in 1999, 2000 and 2001 available at http://www.sei.cmu.edu.) According to one multi-organization statistical study (Clark, 1999), total development costs decreased by 5–10 percent for every further level attained. Another study (Harter et al., 2000) examined 30 software projects in the systems integration division of a large IT firm over the period 1984–96, and estimated the effects of moving from Level 1 to Level 3 to be an increase of 578 percent in the lines of code per error, a reduction of 30 percent in cycle time, and a reduction of 17 percent in person-months of effort. (Other multi-organization studies include Krishnan et al., 2000; Herbsleb et al., 1997.)

8 Engeström (1987) uses the phrase 'expanding the object' to convey the idea of broadening the range of features and goals addressed in the subject's practical engagement with the object.

9 In offering software development organizations a prescription for their future that was based on lessons drawn from the industry's past, the CMM functioned in the 'proleptic' manner described by Cole (1996: 183 ff.). More metaphorically, the CMM can be said to have offered 'scaffolding' for the software process improvement task. The metaphor of scaffolding, originally articulated by Wood et al. (1976), refers to the temporary assistance provided by teachers/adults to students/children as they strive to accomplish a task in their 'zone of proximal development' (Vygotsky, 1962, 1978; Griffin and Cole, 1984).

10 This concern echoed critiques of the 'top–down' nature of the scaffolding metaphor (Stone, 1993; Butler, 1998).

11 I distinguish three aspects of organization—division of labor, community, and rules—following Engeström's analysis of activity systems. For a more detailed analysis of the complete activity system using all the elements of Engeström's model, see Adler (2004).

12 Mainstream organization theory interprets such inter-group rivalries as a natural tendency of large, heterogeneous groups to break into smaller, more homogeneous ones. It has no explanation for why such centrifugal forces prevail so commonly over the centripetal forces created by broader social identities. The answer becomes clearer once we reframe the question in the context of capitalist firms: here, top managers function as agents of owners rather than of workers, and this principle of upward accountability penetrates down into the management hierarchy, pitting subunits against each other as their managers compete for private gains.

References

Adler, P. S. (1999) 'Building Better Bureaucracies', *Academy of Management Executive* 13(4): 36–47.

Adler, P. S. (2004) 'The Discipline of Process: The Transformation of Software Development', unpublished ms.

Adler, P. S. and Borys, B. (1996) 'Two Types of Bureaucracy: Enabling and Coercive', *Administrative Science Quarterly* 41(1): 61–89.

Alevsson, M. and Kärreman, D. (2004) 'Interfaces of Control. Technocratic and Socio-Ideological Control in a Global Management Consultancy Firm', *Accounting, Organizations and Society* 29: 423–444

Bach, J. (1994) 'The Immaturity of CMM', *American Programmer* 7(9), at http://www.satisfice.com/articles/cmm.htm.

Bach, J. (1995) 'Enough about Process: What We Need Are Heroes', *IEEE Software* 12(2): 96–8.

Baldwin, C. Y. and Clark, K. B. (2000) *Design Rules, Vol. 1: The Power of Modularity*. Cambridge, MA: MIT Press.

Barley, S. R. and Orr, J., eds (1997) *Between Craft and Science: Technical Work in the United States*. Ithaca, NY: ILR Press.

Bart, C. K. (1999) 'Controlling New Products: A Contingency Approach', *International Journal of Technology Management* 18(5–8): 395–413.

Bhaskar, R. (1993) *Dialectics: The Pulse of Freedom*. London: Verso.

Blackler, F. (1993) 'Knowledge and the Theory of Organizations: Organizations as Activity Systems and the Reframing of Management', *Journal of Management Studies* 30(6): 863–84.

Boehm, B. and Turner, R. (2003) *Balancing Agility and Discipline*. Boston: Addison-Wesley.

Bollinger, T. and McGowan, C. (1991) 'A Critical Look at Software Capability Evaluations', *IEEE Software* 8(4): 25–41.

Bottomore, T., ed. (1983) *A Dictionary of Marxist Thought*. Cambridge, MA: Harvard University Press.

Brunsson, N., Jacobsson, B. and associates (2000) *A World of Standards*. Oxford: Oxford University Press.

Burns, T. and Stalker, G. (1961) *The Management of Innovation*. London: Tavistock.

Butler, D. L. (1998) 'In Search of the Architecture of Learning: A Commentary on Scaffolding as a Metaphor for Instructional Interactions', *Journal of Learning Disabilities* 31(4): 374–85.

Butler, K. and Lipke, W. (2000) 'Software Process Achievement at Tinker Air Force Base, Oklahoma', CMU/SEI-2000-TR-014.

Cardinal, L. B. (2001) 'Technological Innovation in the Pharmaceutical Industry: The Use of Organizational Control in Managing Research and Development', *Organization Science* 12(1): 19–36.

Chaiklin, S., Hedergaard, M. and Jensen, U. J., eds (1999) *Activity Theory and Social Practice*. Aarhus: Aarhus University Press.

Clark, B. (1999) 'Effects of Process Maturity on Development Effort', unpublished paper available at http://sunset.usc.edu/~bkclark/Research.

Clarke, K. B. and Fujimoto, T. (1991) *Product Development Performance*. Boston, MA: Harvard Business School Press.

Cole, M. (1996) *Cultural Psychology: A Once and Future Discipline*. Cambridge, MA: Belknap/Harvard University Press.

Conradi, R. and Fuggetta, A. (2002) 'Improving Software Process Improvement', *IEEE Software* July/August: 92–9.

Craig, T. (1995) 'Achieving Innovation through Bureaucracy: Lessons from the Japanese Brewing Industry', *California Management Review* 38(1): 8–36.

Crocca, W. T. (1992) 'Review of "Japan's Software Factories: A Challenge to U.S. Management"', *Administrative Science Quarterly* 37(4): 670–4.

Crosby, P. B. (1979) *Quality Is Free*. New York: McGraw-Hill.

Crosstalk (1999) Special issue on 'CMM Level 5 at the Ogden Air Logistics Center', 12(5).

Cusumano, M. A. (1991) *Japan's Software Factories: A Challenge to U.S. Management*. New York: Oxford University Press.

Cusumano, M. A. and Shelby, R. W. (1995) *Microsoft Secrets*. New York: Free Press.

Davila, T. (2000) 'An Empirical Study on the Drivers of Management Control Systems' Design in New Product Development', *Accounting, Organizations and Society* 25: 383–409.

DeMarco, T. and Lister, T. (1987) *Peopleware: Productive Projects and Teams*. New York: Dorset.

Diaz, M. and King, J. (2002) 'How CMM Impacts Quality, Productivity, Rework, and the Bottom Line', *Crosstalk*, March, at http://www.stsc.hill.af.mil/crosstalk/2002/03/diaz.html.

Diaz, M. and Sligo, J. (1997) 'How Software Process Improvement Helped Motorola', *IEEE Software* 14(5): 75–81.

Dutta, S. and Van Wassenhove, L. (1998) 'Motorola India and Excellence: Life beyond CMM Level 5', INSEAD case study, unpublished.

Engeström, Y. (1987) *Learning by Expanding: An Activity-theoretical Approach to Developmental Research*. Helsinki: Orienta-Konsultit.

Engeström, Y. (1990a) 'Constructing the Object in the Work Activity of Primary Care Physicians', in Y. Engeström *Learning, Working and Imagining: Twelve Studies in Activity Theory*. Helsinki: Orienta-Konsultit.

Engeström, Y. (1990b) *Learning, Working and Imagining: Twelve Studies in Activity Theory*. Helsinki: Orienta-Konsultit.

Engeström, Y., Miettinin, R. and Punamaki, R.-L., eds (1999) *Perspectives on Activity Theory*. Cambridge: Cambridge University Press.

Foot, K. A. (2002) 'Pursuing an Evolving Object: Object Formation and Identification in a Conflict Monitoring Network', *Mind, Culture and Activity* 9: 132–49.

Friedman, A. L. and Cornford, D. S. (1989) *Computer Systems Development: History, Organization and Implementation*. Chichester: John Wiley.

Galbraith, J. R. (1977) *Organization Design*. Reading, MA: Addison-Wesley.

Gibbs, G. G. (1994) 'Software's Chronic Crisis', *Scientific American*, September: 86–92.

Gouldner, A. W. (1954) *Patterns of Industrial Bureaucracy*. New York: Free Press.

Greenbaum, J. M. (1979) *In the Name of Efficiency*. Philadelphia, PA: Temple University Press.

Griffin, A. and Hauser, J. R. (1992) 'Patterns of Communication among Marketing, Engineering and Manufacturing: A Comparison between Two New Product Teams', *Management Science* 38(3): 360–73.

Griffin, P. and Cole, M. (1984) 'Current Activity for the Future: The Zo-Ped', in B. Rogoff and J. V. Wertsch (eds) *Children's Learning in the 'Zone of Proximal Development'*, New Directions for Child Development No. 23, pp. 45–63. San Francisco: Jossey-Bass.

Griss, M. L. (1993) 'Software Reuse: From Library to Factory', *IBM Systems Journal* 32(4): 548–66.

Hall, R. H. (2001) *Organizations: Structures, Process, and Outcomes*, 8th edn. Upper Saddle River, NJ: Prentice-Hall.

Harris, R. D. (2004) 'Organizational Task Environments: An Evaluation of Convergent and Discriminant Validity', *Journal of Management Studies* 41(5): 857–82.

Harter, D. E., Krishnan, M. S. and Slaughter, S. A. (2000) 'Effects of Process Maturity on Quality, Cost and Cycle Time in Software Product Development', *Management Science* 46(4): 451–66.

Herbsleb, J., Zubrow, D., Goldenson, D., Hayes, W. and Paulk, M. (1997) 'Software Quality and the Capability Maturity Model', *Communication of the ACM* 40(6): 30–40.

Holt, G. R. and Morris, A. W. (1993) 'Activity Theory and the Analysis of Organizations', *Human Organization* 52(1): 97–109.

Humphrey, W. S. (2002) 'Three Process Perspectives: Organizations, Teams, and People', *Annals of Software Engineering* 14: 39–72.

Humphrey, W. S., Snyder, T. R. and Willis, R. R. (1991) 'Software Process Improvement at Hughes Aircraft', *IEEE Software* 8(4): 11–23.

Ilyenkov, E. V. (1982) *The Dialectics of the Abstract and Concrete in Marx's Capital*. Moscow: Progress Publishers.

Jelinek, M. and Schoonhoven, C. B. (1993) *The Innovation Marathon*. San Francisco: Jossey-Bass.

Jones, C. (2002) 'Defense Software Development in Evolution', *Crosstalk*, November: 26–9.

Keeni, G. (2000) 'The Evolution of Quality Processes at Tata Consultancy Services', *IEEE Software*, July/August: 79–88.

Kirsch, L. J. (1996) 'The Management of Complex Tasks in Organizations: Controlling the Systems Development Process', *Organization Science* 7(1): 1–21.

Kraft, P. (1977) *Programmers and Managers: The Routinization of Computer Programming in the United States*. New York: Springer Verlag.

Krishnan, M. S., Kriebel, C. H., Kekre, S. and Mukhopadhyay, T. (2000) 'Productivity and Quality in Software Products', *Management Science* 46(6): 745–59.

Lawrence, P. R. and Lorsch, J. W. (1967) *Organization and Environment: Managing Differentiation and Integration*. Boston, MA: Harvard University Graduate School of Business Administration.

Leont'ev, A. N. (1978) *Activity, Consciousness, and Personality*. Englewood Cliffs, NJ: Prentice-Hall.

Lieberman, H. and Fry, C. (2001) 'Will Software Ever Work?' *Communications of the ACM* 44(3): 122–4.

Lynn, L. H. (1991) 'Assembly Line Software Development', *Sloan Management Review* 32(4): 88–91.

McGarry, F. and Decker, W. (2002) 'Attaining Level 5 in CMM Process Maturity', *IEEE Software*, November/December: 87–96.

Marx, K. (1845/2002) 'Theses on Feuerbach', trans. Cyril Smith, at http://www.marxists.org/archive/marx/works/1845/theses/index.htm, accessed 21 September 2004.

Marx, K. (1977) *Capital*, vol. 1. New York: Vintage.

Merchant, K. (1998) *Modern Management Control Systems*. Upper Saddle River, NJ: Prentice-Hall.

Meyer, J. W. and Rowan, B. (1997) 'Institutionalized Organizations: Formal Structure as Myth and Ceremony', *American Journal of Sociology* 83: 340–63.

Meyer, J. W. et al. (1983) ' Institutional and Technical Sources of Organizational Structure', in John W. Meyer and W. Richard Scott (eds) *Organizational Environments*, pp. 45–6. Beverly Hills, CA: Sage.

Mintzberg, H. (1979) *The Structuring of Organizations: A Synthesis of the Research*. Englewood Cliffs, NJ: Prentice-Hall.

Nardi, B. A. (1996) 'Studying Context: A Comparison of Activity Theory, Situated Action Models, and Distributed Cognition', in B. A. Nardi (ed.) *Context and Consciousness: Activity Theory and Human–Computer Interaction*, pp. 69–102. Cambridge, MA: MIT Press.

Ngwenyama, O. and Nielson, P. A. (2003) 'Competing Values in Software Process Improvement: An Assumption Analysis of CMM from an Organizational Culture Perspective', *IEEE Transactions on Engineering Management* 50(1): 100–12.

Nixon, B. (1998) 'Research and Development Performance Measurement: A Case Study', *Management Accounting Research* 9: 329–55.

Organ, D. W. and Green, C. N. (1981) 'The Effects of Formalization on Professional Involvement: A Compensatory Process Approach', *Administrative Science Quarterly* 26: 237–52.

Ouchi, W. (1979) 'A Conceptual Framework for the Design of Organizational Control Mechanisms', *Management Science* 25(9): 833–48.

Paulk, M. C., Curtis, B., Chrissis, M. B. and Weber, C.V. (1993a) 'Capability Maturity Model for Software, Version 1.1', Software Engineering Institute, CMU/SEI-93-TR-24, DTIC Number ADA263403.

Paulk, M. C., Weber, C. V., Garcia, S. M., Chrissis, M. B. and Bush, M. W. (1993b) 'Key Practices of the Capability Maturity Model, Version 1.1', Software Engineering Institute, CMU/SEI-93-TR-25, DTIC Number ADA263432.

Pitterman, B. (2000) 'Telecordia Technologies: The Journey to High Maturity', *IEEE Software*, July/August: 89–96.

Podsakoff, P. M., Williams, L. J. and Todor, W. T. (1986) 'Effects of Organizational Formalization on Alienation of Professionals and Non-professionals', *Academy of Management Journal* 29: 820–31.

Powell, W. and DiMaggio, P., eds (1991) *The New Institutionalism in Organizational Analysis*. Chicago: University of Chicago Press.

Raelin, J. A. (1985) 'The Basis of Professionals' Resistance to Management Control', *Human Resource Management* 24(2): 147–75.

Redding, P. (2002) 'Georg Wilhelm Friedrich Hegel', *The Stanford Encyclopedia of Philosophy (Summer 2002 Edition)*, ed. Edward N. Zalta, at http://plato.stanford.edu/archives/sum2002/entries/hegel/.

Saiedian, H. and Kozura, R. (1995) 'SEI Capability Maturity Model's Impact on Contractors', *IEEE Computer* 28(1): 16–26.

Schumpeter, J. (1942/1976) *Capitalism, Socialism and Democracy*. New York: Harper.

Scott, W. R. (1995) *Institutions and Organizations*. Thousand Oaks, CA: Sage.

Scott, W. R. (2003) *Organizations: Rational, Natural, and Open*, 5th edn. Upper Saddle River, NJ: Prentice-Hall.

Software Engineering Institute (2004) 'Process Maturity Profile of the Software Community, 2004 Mid-year Update', at http://www.sei.cmu.edu.

Standish Group (1994) *The Chaos Report*, at http://www.standishgroup.com.

Stone, C. A. (1993) 'What Is Missing in the Metaphor of Scaffolding?', in E. A. Forman, N. Minick and C. A. Stone (eds) *Contexts for Learning: Sociocultural Dynamics in Children's Development*, pp. 169–83. New York: Oxford University Press.

Swanson, K., McComb, D., Smith, J. and McCubbrey, D. (1991) 'The Application Software Factory: Applying Total Quality Techniques to Systems Development', *MIS Quarterly*, December: 567–79.

Thompson, J. D. (1967) *Organizations in Action*. New York: McGraw Hill.

Thompson, P. (1989) *The Nature of Work*, 2nd edn. London: Macmillan.

Vygostky, L. S. (1962) *Thought and Language*. Cambridge, MA: MIT Press.

Vygotsky, L. S. (1978) *Mind in Society*. Cambridge, MA: Harvard University Press.

Weber, H., ed. (1997) *The Software Factory Challenge*. Amsterdam: IOS Press.

Wertsch, J. V., ed. (1979) *The Concept of Activity in Soviet Psychology*. Armonk, NY: M. E. Sharpe.

Westphal, J., Gulati, R. and Shortell, S. M. (1997) 'Customization or Conformity? An Institutional and Network Perspective on the Content and Consequences of TQM Adoption', *Administrative Science Quarterly* 42: 366–94.

Wheelwright, S. C. and Clark, K. B. (1992) *Revolutionizing Product Development*. New York: Free Press.

Wigle, G. B. and Yamamura, G. (1999) 'SEI CMM Level 5: Boeing Space Transportation Systems Software', in G. G. Schulmeyer and J. I. McManus (eds) *The Handbook of Software Quality Assurance*, 3rd edn, pp. 351–80. Upper Saddle River, NJ: Prentice-Hall.

Willis, R. R., Rova, R. M., Scott, M. D., Johnson, M. I., Moon, J. A., Shumate, K. C. and Winfield, T. O. (1998) 'Hughes Aircraft's Widespread Deployment of a Continuously Improving Software Process', SEI Technical Report CMU-SEI-98-TR-006.

Wood, D., Bruner, J. and Ross, G. (1976) 'The Role of Tutoring in Problem-Solving', *Journal of Child Psychiatry and Psychology* 17: 89–100.

Zigura, I. and Buckland, B. K. (1998) 'A Theory of Task/Technology Fit and Group Support System Effectiveness', *MIS Quarterly* 22(3): 313–34.

Paul S. Adler is Professor of Management and Organization at the Marshall School of Business, University of Southern California. His research focuses on organization theory and design in contexts such as banking, engineering, automobile assembly, software, and health care. **Address:** M&O Dept, Marshall School of Business, University of Southern California, Los Angeles, CA 90089–0808, USA. [email: padler@usc.edu]