5

Beyond Hacker Idiocy

The Changing Nature of Software Community and Identity

Paul S. Adler

Introduction

Professions are often taken as prototypical of a more collegial form of community, one that can be contrasted with both traditional *Gemeinschaft* and modern *Gesellschaft*. Indeed, the collegiality of professionals doing intrinsically meaningful work stands as a prefigurative model of the communist utopia of a 'free association of producers.'[1] The key features of professions giving them this status are their common professional socialization and their autonomy in economic status and in technical decision making.

In reality however, professionals' autonomy is increasingly restricted. First, a substantial and growing proportion of professionals work under heteronomous conditions, as employees of capitalist firms: engineers are the largest category of these 'organizational professionals.' Even without economic autonomy, professionals traditionally enjoyed considerable technical autonomy; but this technical autonomy too has been undermined by recent trends towards the bureaucratization of professional work.[2]

Many observers interpret the loss of economic and technical autonomy as a sign of abject proletarianization.[3] If such is indeed the future of professional work, the plausibility of the central thesis of this volume would be seriously undermined. After all, professions are the occupational category most directly implicated in the trends towards knowledge-intensive and solutions-oriented production that we have interpreted as driving the movement towards collaborative community.

This chapter challenges that pessimistic prognosis: I argue that it is precisely the loss of autonomy that is allowing professionals to move beyond the primitive, guild form of coordination towards a more advanced form—collaborative community. To ground my argument, this chapter examines the case of software developers.

Software is a fruitful field for this investigation. On the one hand, in the early years of the industry programmers—much like the physicians described by Maccoby in Chapter 6—resembled guild craftsmen, sharing elements of common socialization and enjoying great technical autonomy in their work. To quote one of them:

> I remember that in the fifties and early sixties I was a 'jack of all trades.' As a programmer I got to deal with the whole process. I would think through a problem, talk to the clients, write my own code, and operate the machine. I loved it—particularly the chance to see something through from beginning to end.[4]

On the other hand, over the past few decades, a significant part of the software industry has undergone massive bureaucratization. A growing proportion of developers describe their work in terms more like these, taken from an interview with a developer who worked in a large systems-development consulting firm I will call GCC:

> 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.

Notwithstanding the development of powerful programming technologies over the recent decades, the guild model common in the industry's early years has proven grossly inadequate for the development of larger, more complex software systems. As systems have grown larger, so too has the proportion of projects that fail to meet their goals or fail entirely, creating what many observers call a state of 'crisis' and 'chaos.'[5] The effort to tame the chaos in software development has led to an increasing focus on ‘process,’ understood as the standardization, formalization, and management control of work processes. One popular vehicle for attaining greater process discipline is the Capability Maturity Model (CMM®) developed by the Software Engineering Institute (described in more detail below). At high levels of ‘maturity,’ software development should, its proponents argue, resemble a bureaucratic factory process in its disciplined operations, predictable results, and continuous improvement. The interviewee from GCC quoted above is from an organization at the highest CMM maturity level.

On the one hand, this bureaucratization promises considerably greater efficiency, timeliness, and quality.[6] On the other hand, concerns have been voiced about the CMM’s bureaucratic nature. These concerns echo those addressed to the broader family of ‘software factory’ concepts of which the CMM is a part.[7] In particular, concern is 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.[8] One software development manager interviewed in the present study expressed the concern 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.’

The debate about the future of software development, like the broader debate about the effects of bureaucratization on professional work, hinges crucially on our understanding of how professionals’ communities and identities are reshaped by bureaucratic rationalization. So far, however, we have little research on these issues. Many observers simply assume that developers will reject this bureaucratization; but my interviews at GCC suggested a more complex picture. The developer quoted above—‘I was kind of shocked’—went on to say:

> 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.

The research reported here analyzes the experience of software developers in four software development units within GCC that have adopted the CMM and reached relatively high levels of maturity. My main findings are threefold: (*a*) under the influence of the CMM, the software development process had become more interdependent, and thus had been 'socialized' in the sense used by Marx and Engels;[9] (*b*) as a result, the structure of development organizations has shifted towards an 'enabling bureaucracy' form, one which combines high levels of community and hierarchy principles of organization, and creating a form of community that is collaborative rather than merely associational or guildlike; (*c*) the subjective identity of developers shifted from the individualistic 'hacker' form towards interdependent self-construal, in the sense used by cultural psychology;[10] and (*d*) these trends were simultaneously stimulated, distorted, and retarded by profitability pressures.

Theoretical starting points

My study takes as its theoretical starting point Marx's analysis of the capitalist production process. As discussed in Chapter 1, Marx saw capitalism as a system characterized by the production of 'commodities'—goods and services produced for sale on the market rather than for direct use by the producers. The commodity is thus a 'contradictory unity' of use-value (utility) and exchange-value (its value in exchange)—contradictory, because the imperatives of making something useful are not necessarily congruent with the imperatives that flow from the desire for profit.[11]

Within the capitalist production process, writes Marx, the basic use-value/exchange-value contradiction is expressed in the contradiction between the *labor process*, in which use-values in the form of working capacity, tools, and materials are combined to create new use-values, and the *valorization process*, in which these use-values appear in the monetary form of wages, fixed capital, and circulating capital, and are combined to create new exchange-value in the form of profit.[12] In Marx's view of the development of capitalism, the contradiction between these two aspects of the production process intensifies over time, as the labor process embodies a tendency towards what Marx called the 'socialization' of the forces of production, while the valorization process embodies the persistence of private-property-based relations of production. Valorization pressures simultaneously encourage, undermine, and distort the tendency towards socialization.[13] Let us unpack this summary formulation.

Socialization is commonly construed as the process whereby people new to a culture internalize its norms: Marx's use is broader. Marx's discussion of the socialization of the forces of production (as distinct from his arguments in favor of the socialization of property relations through nationalizations) suggests that this psychological internalization is just one form of a more general phenomenon: the forces of production are socialized insofar as they come to embody the capabilities developed in the larger society rather than only those that emerge from isolated, local contexts.[14]

The 'objective' socialization of the forces of production is visible at the societal level in the increasingly complex social division of labor—the specialization of industries and regions, and their increasing global interdependence.[15] At the enterprise level—where society's forces of production are instantiated as specific labor processes and specific collectivities—objective socialization was characterized by Engels in these terms:[16]

> Before capitalist production. i.e. in the Middle Ages. . . . the instruments of labor—land, agricultural implements, the workshop, the tool—were the instruments of labor of single individuals, adapted for the use of one worker . . . [The bourgeoisie transformed these productive forces] from means of production of the individual into *social* means of production, workable only by a community of men. The spinning-wheel, the hand-loom, the blacksmith's hammer were replaced by the spinning-machine, the power-loom, the steam-hammer; the individual workshop, by the factory, implying the cooperation of hundreds and thousands of workmen. In like manner, production itself changed from a series of individual into a series of social acts.

Firms develop a whole panoply of management techniques to master what Marx calls the 'cooperation' necessary to coordinate this interdependent 'series of social acts.'[17] In this light, the emergence of bureaucracy can be seen as a key part of the socialization process.

To these objective dimensions of socialization corresponds a subjective dimension—to reprise the conventional meaning of socialization. When the effective subject of production is no longer an individual worker but the 'collective worker,'[18] workers' identities change—workers are resocialized. Socialization in this subjective sense can be understood as the emergence of more 'interdependent self-construals.'[19] The civilizing mission of capitalism is not only to stimulate enormously the quantitative development of the objective components of the forces of production, but also to take a decisive step in the realization of humankind's fundamentally social nature: 'When the worker cooperates in a planned way with others, he strips off the fetters of his individuality, and develops the capabilities of his species.'[20]

The development of the forces of production pulls workers out of what Marx and Engels call 'rural idiocy' and 'craft idiocy.'[21] Marx's use of the term idiocy preserves both its colloquial sense and the meaning from the Greek *idiotes*, denoting an asocial individual isolated from the polis. At the opposite end of the spectrum from the *idiotes*—in the form of the farmer or the craftsman—is the 'social individual,' described in the *Grundrisse* as the technician who accesses and deploys society's accumulated scientific and technological knowledge in a collaboratively organized production process.[22]

Under capitalism, this socialization tendency is simultaneously stimulated, retarded, and distorted by the prevailing relations of production. Competitive pressures force firms to break down parochialisms, to bring everyone into the world market, and to stimulate technological progress; but instead of a broadening association of producers progressively mastering their collective future, capitalism imposes the coercion of quasi-natural laws of the market over firms and the despotism of corporate bureaucracy over workers. The limitations on collective mastery that result from the dominance of the market over firms are visible in capitalism's inability to manage public goods and externalities. The limitations resulting from the coercion of the market are visible in the difficulties facing firms that attempt to establish collaborative relations with other firms up- and down-stream, only to find that competitive pressure destroys these high-trust relations. The limitations resulting from the despotic authority of managers over workers within firms is visible in the Sisyphean nature of corporate human resource management strategies—condemned to futility by the capitalist firm's need for workers who are simultaneously dependable and disposable.[23] With the increasing complexity of technology and the growing knowledge intensity of the economy, these handicaps become increasingly intolerable fetters on social development.[24]

In the overall dynamics of capitalism, these various constraints must and do slowly cede to the overall progress of socialization. In modern industry, competitive advantage often flows from skill upgrading and from greater collaborative interdependence within and between firms. The pursuit of those sources of competitive advantage makes capitalists the 'involuntary promoters' of socialization.[25] While at first, socialization appears in the alienated form of coercive market and bureaucratic control, the subsequent development of capitalism constantly brings forth new, more conscious and collaborative forms of socialization—forms which stand in ever-sharper contrast to the prerogatives of capital.

As a result of the persistence of capitalist relations of production, the path of socialization—the emergence of collaborative community—is halting and uneven. Globalization integrates markets, but by whipsawing regions against each other. Management mobilizes the collective worker, but then is seduced by the easy profits from downsizing and outsourcing. There is a long-term skill upgrading trend, but firms find the low road of deskilling and super-exploitation ever tempting.

The body of this chapter shows that this dialectic of socialization and valorization drives the evolution of software development and the mutations of professional communities and identities. First, however, I describe the context of the research and my research methods.

Some context

The Capability Maturity Model

In the 1980s, the US Air Force studied seventeen major software systems contracts and found that every project was late (by an average of 75 per cent) and over budget.[26] The chaos in large-scale commercial sector projects was (and still is) in general even worse.[27] 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 process maturity framework, releasing a preliminary description in 1987 and the first official version (version 1.1) in 1991.

A broad community of industry people helped shape the CMM. One of the CMM authors writes: 'Nearly 1000 external reviewers who were part of a "CMM Correspondence Group" had the opportunity to comment on the various drafts leading to CMM version 1.1. A CMM Advisory Board helped the SEI review and reconcile conflicting requests.'[28] The software CMM was subsequently complemented by CMM tools for systems engineering, people management, software acquisition, and engineering. 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 5.1. The

Table 5.1. The Capability Maturity Model

Level	Focus and description	Key process areas	Distribution of appraised organizations in: 1987–1991 (132 organizations)	2000–mid-2004 (1,543 organizations)
Level 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.		80.0%	9.6%
Level 2: Repeatable	**Program management processes:** Basic program management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on programs with similar applications.	• software configuration management • software quality assurance • software project tracking and oversight • software project planning • requirements management	12.3%	42.6%
Level 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 programs 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	6.9%	30.1%

Table 5.1. The Capability Maturity Model (*Cont'd*)

Level	Focus and description	Key process areas	Distribution of appraised organizations in: 1987–1991 (132 organizations)	2000–mid-2004 (1,543 organizations)
Level 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 • process change management	0.0%	8.6%
Level 5: Optimizing	**Continuous process improvement:** Improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.	• technology change management • defect prevention	0.8%	9.6%

Sources: Paulk et al. (1993); Software Engineering Institute (2004).

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—it corresponds to the traditional model of professional work. Level 2 represents the rationalization of the management of individual projects. Level 3 characterizes the way the organization manages its portfolio of projects. Level 4 addresses the quantification of the development process. Level 5 addresses the continuous improvement of that process. The underlying philosophy of this hierarchy was inspired by Crosby's five stages of TQM maturity—uncertainty, awakening, enlightenment, wisdom, and certainty.[29]

Early CMM assessments revealed a startlingly 'immature' state of software process, but one that conforms to the expectations of mainstream contingency theory: fully 80 per cent of the 132 organizations assessed during 1987–91 were found to be at the ad hoc Level 1. Over the subsequent years, there appears to have been significant shift (although it is difficult to tell given the changing and unrepresentative nature of the sample). 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 pre-packaged software products.)

Evidence is slowly accumulating that moving up the CMM hierarchy leads to improvements in product cost, quality, and timeliness. The SEI website lists several case studies of high-maturity organizations and the benefits they have achieved. According to one multi-organization statistical study, total development costs decreased by 5 to 10 per cent for every further level attained.[30] Another study examined thirty 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 3 to be an increase of 578 per cent in the lines of code per error, a reduction of 30 per cent in cycle time, and a reduction of 17 per cent in person-months of effort.[31]

Skeptics remain unconvinced.[32] Critics of process approaches argue that these gains may be specific to the sampled organizations. More fundamentally, they may be earned at the expense of developer morale and commitment, and given the importance of developers' attitudes to performance, any performance gains may therefore be ephemeral.

However, we currently lack any reliable evidence on how developers actually respond to the discipline of CMM. Part of the problem is the inadequacy of most of the theoretical framings available for such research. The most common framing assumes that autonomy is a primordial psychological need—especially for professionals like software developers.[33] On reflection, however, it would beg the question to take autonomy as a key variable. It is surely insufficient to characterize the bureaucratization of software work only by what is lost: we also need to understand what replaces that lost autonomy. Abstractly speaking, autonomy is replaced by greater task interdependence and correspondingly more intensive coordination efforts. Autonomy theories assume that coordination is coercive, and that autonomy is thus replaced by dependence and domination. But this assumption needs to be tested against reality, since it is possible that autonomy is replaced by a more symmetrical and collaborative form of coordination—one that might be experienced very differently by employees.

Research methods

The research was conducted in a large, US-based professional service firm, which I will call GCC. With the support of senior management, interviews were conducted with staff in four programs in GCC's Government Systems group. ('Programs' at GCC were organizational units devoted to long-standing, multi-project client engagements.) The choice of these programs was guided by a research strategy that sought to understand how development staff experienced work at high CMM Levels. At the time of my research, Program A was at CMM Level 5; Program A's sister, Program B, was Level 3; Program C was almost Level 5 (it was certified Level 5 shortly after my study); and Program C's sister, Program D, was at Level 3.

In late 1999, I interviewed between fifteen and twenty-two people at various hierarchical levels and in various functions 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 relevant excerpts 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.

To bring some order to these materials, I moved back and forth between existing theories and concepts derived inductively from the interview data. A draft report was circulated to interviewees and other interested parties at GCC, and their comments and corrections were incorporated in subsequent iterations. The present article distills some of the key arguments.

Building process at GCC

GCC is one of the largest software services firms in the world. Major players in this industry include Accenture, IBM, EDS, and CSC. In 2000, GCC's total sales exceeded $9 billion. It employed around 58,000 people. GCC had experienced double-digit annual revenue growth over most of the prior decade.[34]

At GCC, as in other large organizations, the term 'process' was used to refer to a whole hierarchy of standard operating procedures, from 'Policies' defining broad, corporate requirements down to 'Instructions' defining individual tasks. The 'granularity' of process at its finest levels can be gauged by the Instructions at one of the Level 5 programs, Program C. Amongst myriad other categories there were separate Instructions that covered high-level design, two types of low-level design, two types of code reviews, one for testing, as well as Instructions for filling out change request implementation forms and root-cause analysis forms. Each Instruction was several pages in length. They often included the specific forms to be completed as well as flow charts detailing the sequence of associated tasks. Overall, the process documentation summed to some eight linear feet of shelf space. A sketch of the hierarchy of the Government Systems group's process documentation is given in Table 5.2. In recent years, almost all of this documentation had been put online, along with a host of other management information and communication tools. Prescribed work-flows were being built into automated document routing systems. Table 5.2 also describes this technology infrastructure.

If the documentation that developers were required to read was voluminous, so too was the documentation that they were required to write. In the words of one interviewee (perhaps exaggerating for dramatic 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 detail 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

Table 5.2. Process documentation and infrastructure

PROCESSES			INFRASTRUCTURE
Policies and procedures were listed under nine headings: ● General ● Organization ● Human resources ● Security and facilities ● Program development ● Legal/contracts ● Process definition and management ● Finance ● Procurement	**Process definition and management** broke down into the following components. These mapped approximately onto the CMM Key Process Areas: ● Process definition and management ● Project planning ● Project tracking ● Requirements management ● Intergroup coordination ● Software acquisition and management and planning ● Subcontract management ● Quality assurance plan ● Configuration management ● Life-cycle engineering ● Training management ● Software quality management ● Quantitative process management ● Defect prevention ● Process change management ● Technology change management	**Project tracking** included: ● Policy statement ● Procedures: - Cost and schedule tracking - Technical metrics tracking - Project reviews - Risk management ● Instructions	1. Lotus Notes database: ● Document library - Program A development methodology - standards and procedures - guidebooks ● Process assets - process improvement initiatives - technology management - reference documents - lessons learned ● Software Measurement System - collected data - data analysis - collection status - tools/help ● ISO/CMM Discussion - deployment team meeting minutes - CMM/ISO briefings - process briefings 2. Automated workflow management tools

signed. ... I used to be an independent developer for about three years. I never even created a flow chart of my work! The only documentation I needed was a 'to do' list. So I had to change a lot of habits when I got here. (B developer)

To some extent, this formalization was due to size. The projects in the programs under study here were not huge (see details below), but were large enough to require a level of formalization and standardization that was noticeably higher than some employees had experienced in prior positions in smaller establishments. Moreover, government clients typically imposed more documentation requirements than commercial sector clients. And in Programs A and C, formalization was further increased due to the life-threatening risks associated with the products that the software was supporting.

These were, of course, differences in the degree of routineness across the programs and across the departments within the programs; however, all the departments in all four programs—from the most routine to the least—had come under great pressure to adopt a more mature process orientation. Partly this was due to pressure to conform to the expectations of key customers, and partly it was because progress towards higher CMM Levels appeared to at least some influential insiders as a valuable opportunity for technical process improvement. As a result of these external and internal pressures, all four programs had considerably increased the degree of standardization and formalization of all the aspects of the development process. The two high-maturity programs (A and C) had gone significantly further than Programs B and D, with detailed Instructions specifying the work of even the development functions.

It is particularly noteworthy that this drive to bureaucratization took place in organizations devoted to relatively innovative tasks. While the degree of innovativeness was not as high as we might find in a pure research environment, it was substantial since each project represented a novel challenge: all these programs delivered fully customized systems as well as maintaining current systems. Moreover, over recent years, even as they were bureaucratizing, the programs were becoming more rather than less innovative. In all four programs, customer requirements were becoming more complex; new technologies and languages were being introduced at an accelerating rate; and the programs were being pushed to show ever-greater flexibility in responding to customer demands. As a result, training budgets had increased in all four programs in recent years. A program manager at Program C noted, 'Our main challenge has been to transition a staff of 400 plus to new languages and technologies. That has engaged us in a large-scale training effort.'

A common human resource management challenge

Software development relies primarily on relatively 'professional' personnel. It is true that software developers' claim to professionalism is limited because they have not established an accreditation monopoly such as that accorded physicians and lawyers, and because their knowledge base is not as theoretical (although a discipline of 'software engineering'—of which CMM is a direct reflection—has begun to change this).[35] Nevertheless, two factors encouraged a professional status and outlook. First, software development depends critically on the capabilities of the staff, and these capabilities are more occupation specific and less firm specific than is the case with less-professionalized occupations. As evidence, I would cite the fact that in the four focal units, two-thirds of the personnel held a bachelor's, master's, or Ph.D. degree. And second, notwithstanding the discipline created by process maturity, developers needed to exercise considerable discretion in their work to ensure quality and efficiency. Developers thus had considerable power in their relations with management. This excerpt illustrates:

> 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)

Given this relatively professional character of the work, and given also the persistent imbalance of supply and demand in the labor market for software personnel, staff retention had long been a high priority for GCC and an important consideration in the management of process rationalization efforts. As government contractors, these GCC programs had little control over salary levels, and salaries tended to be lower than in the commercial sector. Moreover, by policy, GCC offered only modest financial incentives to staff below senior management levels; symbolic rewards were more common. So managers understood that retaining talented employees depended above all on making GCC a good place to work, with both good employment conditions and challenging tasks. Program D illustrates the resulting climate:

> The turnover figures don't show you how many people come back. We've had six or seven people leave then return. They realized that we have a pretty good environment. You don't get crucified for mistakes, for one thing. The 'blame game' is pretty

much non-existent. If someone underperforms, we usually find another job for them that's a better fit. They aren't humiliated. (D department manager)

In the programs studied, annual staff turnover in recent years had been significantly lower than the industry average.

Key findings

To analyze the evolution of software development, we need a theoretical framework that allows us to recognize the mutual influence of community, identity, and the other factors internal and external to the collectivity that affect the organization of software development. To this end, this chapter relies on cultural-historical activity theory (CHAT). CHAT's key ideas are summarized in Appendix 5.2. We take from CHAT the framework summarized in Fig. 5.1.

In the sections below, I will discuss each of the elements of the software development activity system in turn. In each case, we will see the contradiction between valorization and socialization at work. By way of introduction, I begin with an overview.

Traditionally, at the lowest levels of process maturity, Level 1, developers enjoyed high levels of autonomy, task variety, and task identity. Recall the interview quoted in the introduction. The activity systems within which developers worked were largely local. As another study of programmers of that period noted, 'Programmers (and analysts) followed a logic and procedures which were largely of their own making.'[36] Being tacit rather than codified, tools and rules were difficult to communicate across locales. Working knowledge was in these senses private rather than social.

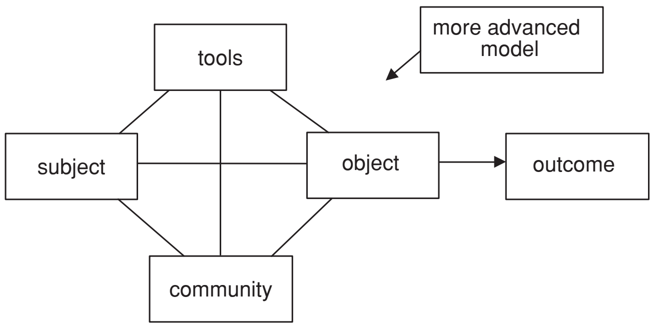

Fig. 5.1 The structure of an activity system (adapted from Engeström 1987)

At higher levels of process maturity, developers were embedded in larger, more complex organizations, and encountered approaches that were the fruit of a complex, organized, large-scale process design and development effort—recall the second interview quoted in the introduction. Tools, models of the development process, and the structuring of the work community were no longer naturally emergent phenomena grounded in local experience. They were formalized, standardized, bureaucratized. Developers were aware that their effectiveness was not only the result of their own individual effort and skill and of informally shared tricks of the trade, but also and increasingly the result of this social, rather than private, accumulation of working knowledge.

At first, this socialization took a form many developers experienced as alienating and coercive. Discussing the Military Standards for software quality assurance that came into force in the mid-1980s, one veteran noted that: '[Military Standard] 2167A was supposed to make coding a no-brainer' (C development manager). In the civilian Program A too, the initial experience with process was top down, oriented to conformance, and 'most managers felt that it was just a matter of ensuring that people were implementing it' (A program manager).

But by the time of my study a decade or more later, the Level 5 programs had pushed the socialization of the production process further, and it had taken a more enabling form. Recall the GCC interview quoted in the Introduction: 'But I can see the need now.' Another developer said:

> I came from a background in industrial process computers and the organization I worked for was much less structured in how they handled all this. The process was basically just define the requirements, write the code, then do a final test. Apart from that, you were basically on your own. Here the processes tell you a lot more about how to do the work. By formalizing things, they make it easier to incorporate lessons-learned a lot faster. Previously, it was more like a 'hand-me-down'—you learned how to do your work with some help from other people on the job, or just by yourself. (B developer)

Overall, my interviews suggest that the bureaucratization represented by CMM-style process maturity had broadly positive effects, both technical and attitudinal. Technically, the more routine tasks in software development were rendered more efficient by standardization and formalization, leaving the non-routine tasks relatively unstructured to allow more creativity in their performance. Moreover, non-routine tasks benefited from the guidance afforded by well-designed process controls, and from the elimination of 'noise' and overload created by unnecessary rework in routine tasks.

Attitudinally, process maturity was experienced by many developers as enabling and empowering rather than coercive and alienating. Process maturity did mean a loss of autonomy. Higher CMM Levels drew people into broader and tighter webs of interdependence, both horizontally and vertically. The individualistic 'hacker' model of software development was displaced by a new understanding of work as a collective endeavor. In that collective endeavor, process discipline, even though a constraint on individual autonomy and a burden on individual productivity, was often seen as a functional necessity in the pursuit of individually and collectively valued goals.

The key to ensuring a positive response to process discipline was extensive participation: As several interviews summarized it, 'People support what they help create.' These organizations had both formalized processes supporting participation and strong normative commitments to participative rather than autocratic styles of leadership. The resulting organizational form corresponded to the 'enabling bureaucracy' type, combining a dense web of rules and a finely differentiated vertical and horizontal division of labor with high levels of trust and collective cohesion—combining the community and hierarchy principles of organization.[37] Thus, not only did community become more salient, but the form of community changed—from the guildlike traditional form of professional community to the collaborative community discussed in this volume's Introduction.

On the other hand, however, as we will see below, the forces at work also reflected valorization pressures, and these pressures simultaneously stimulated and limited socialization. Socialization appeared to be the dominant tendency; but the progress of socialization was hesitant and uneven. In each of the following subsections, I first identify the dominant tendency and then one or more accompanying tensions that reflect the various contradictions at work.

Object: an expanded object

Common English usage captures the dual nature of the object of work—both raw materials and purpose. My interviews suggest, first, that process maturity made the object of developers' work more stable and more intelligible. This object was less likely to mutate in unpredictable ways due to the poor quality of configuration control, requirements planning, or quality management:

Before I came to GCC, I worked for a 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 testing)

Second, process meant that the object of developers' work expanded to include more paperwork. Process forced developers to document their work more thoroughly—to attach to the code or the test a description of its meaning, its intent, and the rationale of the design. The developer quoted immediately above went on to say: 'On the other hand, I don't like all the paperwork. Your work may be more streamlined, but after the work is done, there's more forms you have to fill out to document what you've done.'

On the one hand, this paperwork could feel like a burden; but on the other hand, it represented welcome expansion of the object of work to include an imaginary dialogue with previous and future developers (a temporal expansion) and with other people who are working on the code (a social expansion): '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 also enabled a technical expansion of the object of work to include process itself. Developers were called upon to participate in process improvement efforts. The process itself thus became an object of their work:

Perhaps the biggest change as we've become more process mature is that it makes everyone more interested in process improvement. Take an example: now I'm working on a new software utility. Top management asked us to evaluate it, to see if we should all use it. So I've been facilitating a series of meetings with all the managers, where everyone is talking about the utilities they are using and the problems they're having. It's been great to see this kind of problem-solving work going on. That's the effect of having a defined technology change management process [a CMM Level 5 KPA]. CMM got this process going for us. (D logistics)

TENSIONS: 'IN THE END, IT'S ALL ABOUT PROFIT AND MEETING SCHEDULES!'

The object of work is never simply an unproblematic, technical given. When the object of work expands to include CMM certification, tensions appear between competing priorities, between short- and long-term goals, and between technical and business needs:

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)

I understand why we need the CMM evaluations. But it's added a lot to the amount of documentation we need and the number of interviews we have to go through. I suppose that in the long term, this documentation might help us improve, but for the developers, it's added a lot of paperwork. (C developer)

Clearly, part of the CMM effort was 'for show,' responding to symbolic legitimacy pressures rather than technical performance pressures. As such, it sometimes led to a decoupling between formal process and daily practice.[38] Comments such as these were common in the two Level 3 programs but less so in the two Level 5 programs:

We do have written processes, but some are not always used consistently. They are not always being used by the developers. They are not always used by the program managers in their regular reviews. (B process engineering, formerly with A)

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

In part, this symbolic/technical tension in the nature of the object reflected a deeper contradiction, between the use-value and exchange-value aspects of the object. Interviewees were often aware that their (use-value-oriented) process improvement efforts were at risk of being overridden by a higher (exchange-value) imperative:

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)

It's hard to convince people that improving the process will help us get or keep business. [Referring to the downsizing of Program A:] We had a world-class process, and look what happened to us! Jobs in an organization like this depend a lot more on the vagaries of contracting than on our process excellence. (A department manager)

The contradiction between use-value and exchange-value was particularly visible to the interviewees in the form of missed opportunities for process improvement:

One key challenge is maintaining buy-in 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 developer)

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)

In sum, the impact on the transformation of the object under mature process conditions was to make developers more conscious of their interdependence with other contributors in attempting to create a good product, even if the organization's profit imperative sometimes undermined those efforts.

Tools: better information

Under a more mature process, many interviewees felt that they had better tools for software development work. First, as we saw earlier, process was a tool that helped clarify the object of development work:

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. (C development manager)

Second, process functioned as a tool providing valued guidance in the work process:

Developers want above all to deliver a great product, and the process helps us do that. What I've learned coming here is the value of a well-thought-out process, rigorously implemented, and continuously improved. It will really improve the quality of the product. In this business, you've got to be exact, and the process ensures that we are. You have to get out of hacker mode! (A developer)

Process maturity assisted in this by creating a common vocabulary:

In a Level 1 organization, one without a common process, even one where there was a lot of goodwill between the functions, they wouldn't have the common vocabulary, or common definitions of key tasks, and everything would be subject to conflicting interpretation, so people would be fumbling in the dark. A common process greatly simplifies things. (C project manager)

Process maturity also provided tools in the form of objectified and collectively accessible memory:

Process gives people access to assets from prior work—for estimation, for standards and procedures, and for lessons learned. In our asset library, we keep the standards and procedures of all our projects, and project managers refer back to these to use as templates. We encourage people to share and borrow. (A quality assurance)

Take for example our internal software delivery procedure. At first, developers thought that this was just more burdensome paperwork. But soon they found it was a great memory system. (B quality assurance)

Overall, process did not appear to hinder creativity—or at least not the forms of creativity needed in these programs. This excerpt expresses the assessments made by several interviewees:

[E]ven 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 A)

Automation of tools was a crucial precondition for process maturity, since developers needed to consult voluminous documentation and circulate work-in-process in a timely manner. GCC had therefore invested considerable resources into building not only a sophisticated 'development environment' for technical tasks but also an integrated suite of databases and tools for the associated management tasks. The software factory was thus highly automated—contrasting with the 'handicraft' or 'manufacturing' models that prevailed in software development in earlier decades.[39]

Alongside this automation, GCC also sought to streamline the remaining human tasks. Testing exemplified this trend:

Over the last ten years, we've refined the test procedures considerably. First, we have better tools. Documentation and reports that used to take two or three days each week to create can now be generated in an hour. Second, we streamlined some of the procedures for some projects. Now we have a generic template, which we can modify to suit the circumstances. We moved from prescribed, detailed test tables to simpler and voluntary guidelines based on historical examples. And with the benefit of experience and analysis, we are collecting more useful information and less of the kinds of information that proved to be not all that useful. (A testing)

TENSIONS: LESS 'PEOPLE DEPENDENCE' MEANS LESS (INDIVIDUAL) POWER

Sophisticated tools such as those offered by mature processes reduce the dependence of the organization on the individual:

When I arrived here, we had a lot of veterans with deep process knowledge. But as we lost those people, we lost their institutional knowledge. That's why I'm trying to document our process. That will make us less people dependent. (D department manager)

Our process makes us less people dependent. And that goes for managers too. We have promoted the three project managers we used to have, and now we have five new project managers. Bringing these new managers up to speed was much easier with a strong process. (C program manager)

While the direct business benefits were obvious, reducing people dependence also reduced the individual's power vis-à-vis the organization. A fear of vulnerability thus lurked in the background:

If you have a good process, then people become like widgets you can stick into it, and everyone knows what their job is. Obviously that's a big advantage for the organization. . . . On the other hand, it also brings some fear for job security. It does make my job as a programmer easier to fill. (B department manager, formerly with A)

This vulnerability was moderated by the favorable labor-market situation of software professionals. Moreover, our discussion of the community will reveal that process gives more influence to subordinates as well as to managers.

TENSIONS: INSUFFICIENT AUTOMATION AND SIMPLIFICATION

Many interviewees argued that insufficient effort had been devoted to developing the tools needed to lighten the burdens of process. Comments such as these were common:

All these forms have a valid purpose, but it takes so long to fill them out that it just doesn't seem very efficient. We really need a lot more automation in doing all this. (B developer)

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 object had expanded in the eyes of managers and developers, but management had not invested the resources needed to upgrade the tools to support the new, expanded tasks. Such tensions are inevitable when the object itself evidences the (primary) contradiction between use-value—great code, well produced—and exchange-value—'In the end, it's all about profit.' Under such conditions, resource investment decisions are inevitably somewhat inconsistent.

In sum, from the use-value point of view, the transformation of tools under mature process conditions not only helped developers in their daily work, but it did so in a way that made their interdependence increasingly visible. On the other hand, from an exchange-value point of view, developers were expendable variable capital and tools were expensive.

Community: beyond my cube

At higher levels of process maturity, the collective nature of software development work became more explicit as did the organizational architecture of this community:

A well-defined process gives you a kind of map of the whole enterprise. (B quality assurance)

The overall process is more intelligible now. All the organization charts, the people, the processes and documents, and the minutes of various groups are on the website. (C program manager)

Moreover, the boundaries of people's everyday reference group—the community with which people identified—broadened to encompass all the functions that contributed to the final product:

Some programmers here used to be very isolated. We had one fellow who just sat in his cube all day from six in the morning till two in the afternoon. Many of us didn't even know his name! But the process here drew him into team meetings and into new conversations. Eventually we even got him helping with training. (B developer)

The Improvement Team's work created a real sense of community. Each week it would be someone else's turn to present their process. Since you knew it would be your turn soon, we all helped each other. And everybody got to see how the rest of the organization functioned. All the data was shared. I compare that to the way the organization functioned a few years ago. One of our big problems was poor communications across the organization and up and down the organization. No one knew anything anyone else was doing. Now we're working in unison. Process makes for a more unified front for the customer. And it makes you feel important, because you're part of the process and you know where you're at in the process. I'm only a tiny part of the process, but I know that what I do is needed for the success of the whole thing. (D logistics)

Interviewee B-13 was one of the senior process staff at Program A before being transferred to Program B to help its process maturity efforts. His assessment reflected longer experience:

At Level 5, everyone feels part of the enterprise—versus feeling very good technically, in what they do, but hazy about their place in the whole business organization—for example, they can't explain the functions of QA. At Level 5, you understand what other people are doing and why. Everyone can discuss and are involved in improvement efforts, not only technical but also process, organizational problems—versus at Level 1, where the only improvements that people can talk about are local and technical. And at Level 5, measurement is a part of life. At worst, people tolerate it. The majority see it as an integral part of their work—versus at Level 1, where measurement is not part of the culture, where it's not seen as having value, and where it's seen as waste, bureaucratic overhead, and people feel 'Just leave me alone.' (B process engineering, formerly with A)

RULES: CONSTRAINTS THAT MAKE SENSE

Process maturity did mean that there were more procedural rules constraining developers in how they did their work, but it also meant that these constraints were largely seen as means by which the efforts of many contributors to the development activity could be coordinated more effectively:

Here, I'm just a small part of a bigger project team. So you don't do anything on your own. It's a collaborative effort. So there has to be a lot of communication between us. And the process is there to ensure that this communication takes place and to structure it. The process helps keep us all in sync. (C developer)

At higher levels of process maturity rules were more numerous, but developers had more opportunity to participate in defining and refining them. As quoted above, interviewees described the cultures of Programs A and C as having become more participative in recent years. In daily practice, rules took an enabling form. Through a formalized 'Tailoring Cycle,' software development standards and procedures ('S&Ps,' of which there were over 100 at Program A) were modified for each project with participation by the developers themselves:

People have to be a part of defining the process. We always say 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)

When S&Ps are chosen for a project, the rule is that they have to be sent out to everyone affected for review. And sometimes we give some pretty negative feedback! I remember I wrote on one draft, 'Hey, you've forgotten to tell us how to get out of bed in the morning and how to brush our teeth!' It was way too detailed and rigid. Those kinds of things get shot down pretty quickly. Over a period of years, people learned how to write procedures that were reasonable for our work environment. . . .

When I managed software development on one of our bigger projects, I asked all our software developers to help me tailor our S&Ps. The GCC people knew the drill, but we also had some other contractors working on this with us . . . and they would say, 'No, just tell me how you want us to do this.' About a year into the project, I remember one of the contractors who had complained the most about this extra work coming to me to thank me, saying, 'If you'd have written these, I would have just ignored them. But since I helped write them, I've felt duty bound to follow them.' (A developer)

The Tailoring Cycle was not the only vehicle for participation in process definition. In Programs C and D, Software Engineering Process Groups (SEPGs) also served this purpose. In recent years, the SEPG at C, but less so at D, had put increasing weight on encouraging suggestions for process improvement from lower-level staff. Moreover, many departments had process improvement teams. Whereas these teams were sparse and temporary in the less mature programs, they were ubiquitous and ongoing in the more mature programs.

DIVISION OF LABOR: DIFFERENTIATION AND INTEGRATION

The division of labor at GCC differentiated various roles and subunits whose complementary capabilities were integrated both by superordinate goals and by strong process discipline. The following excerpt presents an assessment that is particularly interesting because the interviewee is a woman whose experience of a relatively mature process was recent:

A more mature process means you go from freedom to do things your own way to being critiqued. It means going from chaos to structure. It's a bit like streetball versus NBA basketball. Streetball is roughhousing, showing off. You play for yourself rather than the team, and you do it for the love of the game. In professional basketball, you're part of a team, and you practice a lot together, doing drills and playing practice games. You aren't doing it just for yourself or even just for your team: there are other people involved—managers, lawyers, agents, advertisers. It's a business, not just a game. You have to take responsibility for other people—your team-mates—and for mentoring other players coming up. (B developer)

Process maturity created both more differentiation and more systematic integration in three dimensions of the division of labor: horizontal, line/staff, and vertical. I take them in turn.

As the process became more mature and disciplined, the horizontal division of labor deepened. The span of integration became correspondingly broader: actors developed relations with a broader set of partners. These relations became tighter: the coordination across groups became more rigorous. And they became more collaborative: mutual indifference or rivalry was replaced by active collaboration. These changes were visible within departments as well as between them:

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 ... now the testers are more involved in system concept definition and requirement definition activities. (A quality assurance)

With process maturity, new staff functions such as Configuration Management and Process Engineering emerged, and new line/staff relations were created. QA exemplifies the trend.[40] In the past, QA was often remote from the daily work of developers, arriving on the scene at the end of the work cycle to inspect the output. QA's role evolved with process maturity to (*a*) a greater focus on process quality rather than only product quality, (*b*) greater responsibility for infusing process rather than only auditing it, and (*c*) a closer and more collaborative relation with the line departments. QA's role in the Tailoring Cycle is a good example:

First, QA sits down with project manager and his management team. I'll ask: what processes do you need? Do they exist? We come up with a process approach for the project. Second, project managers work with Process Engineering to resolve the action items out of the first step: what new S&Ps have to be written? Which have to be modified? The project managers, often assisted by their leads, then define the S&Ps they need. Third, this proposal comes to the CRB [Change Review Board] for discussion and approval. Fourth, we try to get the managers of each project to do the training for their S&Ps. Fifth, QA conducts a regular in-progress process audit to check the project's compliance with the process approach they've chosen. And there's also an End-of-life-cycle-phase audit. . . . QA is not a policeman! QA is there to help the project identify the processes you need, tailor existing ones to your needs, learn that process, and do a check to see if you're using it. If I find a problem, it's my job to help the project work out how to address it and how I can help. (B quality assurance)

Finally, in the vertical dimension too, relations grew denser and more collaborative. Process brought greater specificity—clarity and detail—to planning and assessing the progress of work: 'With a more mature process, my manager has visibility into how I do my work and can challenge me on it—I can't just play excuses and he can't use brute force' (B developer).

Process also provided superior–subordinate relations with objective points of reference outside the dyadic authority relationship. Process thus reduced the 'people dependence' of these authority relations just as it reduced people dependence in technical relations. Several interviewees argued that the objective character of the data created in a more mature process gave the subordinate more power. This excerpt illustrates:

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 program manager)

TRAINING: BEYOND APPRENTICESHIP

The acquisition of professional knowledge was rationalized. Process encouraged a shift from a traditional form of training—apprenticeship—towards something more systematic. Apprenticeship is a mode of learning that is appropriate and necessary when knowledge is the local, tacit, private property of the guild artisan-craftsman.[41] A more socialized production process relies on forms of knowledge that are more codified and on forms of training that can thus be more rationalized. Going back a couple of decades, this transformation began with the shift to formal university training requirements for development jobs; more recently, under the pressure of CMM, it continued in the further rationalization of the acquisition of firm-specific skills:

We've developed a formal mentoring program. There's a checklist of the key processes everyone needs to understand, and every new person is assigned a mentor whose job it is to explain each of these in turn. The checklist is audited by QA. (A testing)

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. (C department manager, formerly with A)

TENSIONS: GENERAL VERSUS PAROCHIAL INTERESTS

Differentiation created local identities, which complicated the horizontal integration task. This excerpt illustrates:

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.' (D process engineer, also works with C)

Staff/line relations were not always easy:

By and large, we haven't had too much difficulty bringing our managers around to this more collaborative approach. . . . 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. (A program manager)

Interviewees also discussed tensions in the vertical dimension: management did not always 'listen,' or if they listened, did not always 'hear.' Compared to the horizontal tensions, the vertical ones reflected a deeper, structural asymmetry of power and authority:

How managers react depends a bit on how you present your suggestion. If you present your idea constructively, they're more likely to react positively. If you come at them with complaints and negative criticism, they don't take it as well. Managers are people too! And sometimes how receptive they are depends on other things going on. For example, if they are under pressure from their bosses to move faster, they may not be very receptive to taking time out to redefine the process. (B developer)

Notwithstanding these tensions, it was striking that in its various dimensions, community appeared to be stronger and more cohesive in the Level 5 programs than in the Level 3 programs. A key factor explaining this result was the more extensive participation in process definition we saw in the discussion of rules, above.

TENSIONS: RULES, ENABLING VERSUS COERCIVE

Some rules appeared to developers as burdensome and coercive: 'The forms are all electronic, but we still have to print them out and sign them. Why all this overhead? The theory is that it will increase quality. But I'm not convinced it really helps quality. Sometimes it seems like it's more for accountability' (B developer).

However, the professional character of the workforce gave developers considerable power to resist coercive rules. Earlier, we saw how developers stopped the adoption of Ada. Another excerpt offers a second illustration:

Whenever you force programmers to do paperwork they don't want to do, they get sloppy. They will invent workarounds just to avoid it. And those workarounds can create problems. For example, if you want to create a new sub-routine versus add to an existing one, you have to write a whole new package. So programmers will go a long way to avoid creating a new sub-routine, even if it means that the quality of the code is affected. (B developer)

The importance of buy-in and the temptation of coercion were evidenced in the firmness with which senior management treats instances of autocratic behavior by managers. Among the several cases cited by interviewees in the various programs, this one was illustrative:

Executive management might impose on me as program manager a CMM rating goal without much dialogue, and they might even have a pretty coercive view of what process discipline consists of. But I can't let that flow downhill from me. We explain to our managers why the rating is important to our future. And once you see the heads nodding, then you have to find the right people for the implementation team—people who aren't going to dictate to the other levels how to proceed. We really can't afford an autocratic style of leadership. The risk of losing critical people is too high. . . . We did have a pretty autocratic manager a while back in our software development organization. He had very strong technical skills and would often make decisions without consulting his staff. We heard a lot of complaints, and we saw some turnover too. But his technical skills made him very valuable to us, so we kept him on even after he offered to resign. We tried to get him to change his style, but he didn't, and eventually, after maybe two years of this, we just had to let him go. It was difficult. And he took a few loyal people with him too. (D program manager)

TENSIONS: 'WE STILL DON'T HAVE THE RESOURCES'

Several interviewees pointed to another persistent tension in the structure of community: the lack of dedicated resources for specialized staff departments. The following quotes were illustrative:

We do ask project teams to do a Lessons Learned report at the end of the project. We post the results on the database. But there's no staff support for the process. (A quality assurance)

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 developer)

(As explained in Appendix 5.1, Program C, like D, did not have a dedicated Process Engineering group.) These concerns echo those relative to resources for better tools.

TENSIONS: COMMUNITY VERSUS CLASS

On the one hand, developers and managers were (and saw themselves as) part of a community, part of a collective endeavor. On the other hand, their interests sometimes put them in opposition to each other. We saw this in the earlier discussions of the conflict over Ada and the organization's interest in avoiding 'people dependence.' While senior management put considerable weight on building a sense of common destiny and community, the battle could never be won once and for all. This excerpt illustrates:

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)

In sum, under mature process conditions developers experienced themselves more strongly connected in their communities; but the effect was sometimes undermined by conflicting interests. Process-mature organizations drew developers into collaborative efforts to define and refine their rules; but managers struggled to preserve this participation against the temptations of coercion. A more mature process augmented both differentiation and integration efforts, and as a result, developers saw themselves as part of a larger, more complex, and more interdependent endeavor, one that was often, but not always, collaborative in nature.

Outcome: collaborating with customers

The outcome of the software development activity system at these GCC programs was a market-mediated exchange, giving software in exchange for fees from a distinct activity system in the customer organization. Process was both influenced by and in turn influenced these relations.

As noted above, government agencies were increasingly making CMM Level 3 an important factor in awarding contracts. Program B's contract included an explicit Level 3 requirement, and customer pressure had been a critical impetus to process efforts in Programs A, C, and D too. For customers and suppliers alike, process maturity held the promise that the risks associated with arm's-length market transactions—the inevitable gap between the producer's supply and the customer's demand—could be moderated by mutual commitment to a process that set expectations and guided collaborative decision making. This drew developers into more conscious collaboration with people in the client organization. Instead of submitting to the conventional wisdom that 'The customer is always right,' GCC staff were encouraged to push back and open a dialogue:

> There's a great focus now on 'accountability' all through the system. We are expected to be more aggressive in pushing back when things are inconsistent with our processes. And that goes down to our project managers. Instead of simply supporting our customer management counterparts, the project managers have to be willing to push back. That's changed the tone of some of our monthly program review meetings with the customer. This culture change goes right down to the staff. In general, we try to buffer the staff from these issues, but if they get instructions that violate our processes, they have to push back too. (C program manager)

TENSIONS: COORDINATION VERSUS MARKET ANARCHY

The fact that the customers were economically independent entities pursuing their own priorities sometimes undercut the cooperation needed to assure a high-quality custom software process:

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 A)

Even at high levels of maturity, the outcome node was the site of contradictions between activity systems. Program C illustrates:

Our customer has been rated a CMM Level 4, but they don't seem to implement their process. For example, in one of our recent projects, the requirements kept changing and the scope kept growing, but the customer wasn't following a disciplined process for controlling these changes and just didn't want to hear about the implications. The requirements kept drifting so much that it was very hard to even regularly update our estimate of the size of the project. They just ignored our concerns for nearly a year. Finally we issued a cost report that showed that we'd need 25% more staff-months. Putting it in dollars finally got their attention. But not before we had wasted a lot of work and time. (C project manager)

In sum, the transformation of the outcome under mature conditions made developers more conscious of their interdependence with clients in creating a good product, even if the competing objectives of organizations and subunits distinct profit objectives sometimes undermined that effort.

A more advanced model: the CMM as scaffolding

In all four programs, the CMM was seen by many as representing a more advanced model of software development work, a model that could guide improvement efforts. The challenge of CMM certification was to 'map' the program's existing practices to the CMM's KPAs. In some cases, existing practices were revealed to be satisfactory, and this mapping was therefore experienced as wasteful burden; but in most cases, the CMM provided guidance—the kind of guidance that builders receive from scaffolding—for ongoing process improvement efforts.[42]

Program C had long worked under Military Standards, so the discipline of the CMM was experienced against that backdrop, but its experience was otherwise representative. This excerpt expands on one quoted earlier:

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 process, and that has helped stimulate improvement. (C training)

The CMM could function effectively as scaffolding—as a model of a more advanced activity system—because, and to the extent that, it was seen as an 'industry-validated approach':

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, 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 struggle against proponents of alternative organizational development scenarios, proponents of the CMM had the advantage of this cultural-historical validation.[43]

TENSIONS: SCAFFOLDING VERSUS LEARNER-CENTERED DEVELOPMENT

Notwithstanding the research evidence cited earlier, interviewees were nuanced in their assessments of the impact of CMM on process effectiveness. 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.[44] 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 A)

As discussed above, part of the difficulty was how to juggle the goals of symbolically valuable certification with technically valuable process improvement. On the upside, external legitimacy pressure could facilitate internal change. However, CMM ratings seemed to play a somewhat different role in the different organizations. According to this same interviewee:

I can see that external evaluations are a very important learning tool. It's just like in college: 90% 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 A)

TENSIONS: IS CMM REALLY A MORE ADVANCED MODEL?

Levels 4 and 5 address organization-wide, as distinct from project-specific, capabilities. Optimistically, one might imagine that once a basic level of discipline was established in the conduct of individual projects, even greater improvement might come from organization-wide cross-project discipline, since this would enable an organization to leverage lessons learned from any one project to the whole portfolio of projects. This would seem to be true of many hardware development improvement efforts.[45] So far, however, the experience of the two Level 5 programs was mixed. At Programs A and C, several interviewees—albeit a minority—assessed their skepticism in these terms:

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 training)

Interviews suggested three possible reasons for this mixed reaction. First, the CMM might have been simply misguided in how it characterized Levels 4 and 5. After all, when the CMM was first elaborated, these levels, particularly Level 5, were essentially hypothetical, since so few software organizations had been shown to function in this manner. (As of 2001, however, there were over 130 organizations certified at Levels 4 or 5.) Second, and alternatively, perhaps Levels 4 and 5 were well conceived, but the hypothesized potential benefits would only materialize with further effort and experience. A third, related, possible reason is that the magnitude of the benefits of cross-project processes is related to the number of projects undertaken, and both Program A and Program C undertook a relatively small number of relatively large projects.

Overall, however, the interviews suggested that the CMM had been a valuable tool for organizational improvement, and in doing so, had drawn developers into more conscious collaborative improvement efforts—within their programs, with clients, and with the community behind the CMM.

Subject: a more interdependent self-construal

My interviews suggest that process maturity led to changes in the subjective identity of developers—towards a more interdependent self-construal. What mattered to these professionals' self-esteem and identity was now not so much their individual efficacy as their collective efficacy.[46] In my interviews, 'we' tended to replace 'I' as the subject of work, because people increasingly saw themselves as part of a collective effort. The ratio of mentions of 'we' to mentions of 'I' in my interview transcripts was 1.83 in Program A and 1.95 in Program C (the two Level 5 programs), and 1.29 in Program B and 1.44 in Program D (the two Level 3 programs).

The proximate cause of this change was the experience by developers of changes in the other nodes of the software production activity system. As argued a century ago by pragmatist philosophers and symbolic interactionist sociologists, and as further articulated by CHAT, the self is not an immutable spirit hovering above the material world, merely influenced, but never fundamentally changed, by an external context. The self is an identity whose contours and contents vary with—indeed, are constituted by—the networks of people and things within which the individual is located. The hacker self reflected the structure of CMM Level 1 activity systems. The effect of process maturity on the software development activity system was to socialize its objective elements and thereby to forge a different kind of self—a broader, more interdependent sense of one's identity. Some excerpts illustrate:

In a small organization doing small projects, you have a lot of flexibility, but there's not much sharing. You're kind of on your own. Here, I'm just a small part of a bigger project team. So you don't do anything on your own. It's a collaborative effort. (C developer)

We used to be a group of hackers. If we'd have had to rebuild a system, we simply wouldn't have been able to do it because we wouldn't have had the documents. We've come a long way from that! Now we function according to a defined process and we collect data on ourselves so we can do defect causal analysis to drive continuous improvement. (A quality assurance)

More concretely, this new self-construal emerged through a mix of adult socialization and attraction-selection-attrition processes.[47] On the effects of the former, we have the testimony quoted earlier of 'But I can see the need now.' In Program D, one interviewee described his experience in these terms:

I was not originally a believer in this process stuff. I remember seeing coding guidelines when I joined the Program D. I just threw them into a corner. But a year later, I found that my code didn't make it through the code checker, and that got me to reconsider. So I went to some CMM training a few years ago—and I've been converted! Most of the developers and leads are being dragged into process kicking and screaming. Any coder would rather just hack. (D process engineer)

On the importance of attraction-selection-attrition, two excerpts are illustrative:

You won't fit in well here if you don't like structure, you prefer working by yourself, you don't like getting suggestions from other people, or you don't like taking responsibility for your work and for making it better. (A project manager)

We still have to deal with the 'free spirits' who don't believe in process. . . . Most of them adapt, although some don't and they leave. (C process engineer)

COSMOPOLITANS VERSUS LOCALS

Professionals tend to have strong ties to and identify with their occupational community: they traditionally have little identification with the local, employer organization.[48] Interviews suggested that the greater process maturity strengthened both developers' cosmopolitan orientations and their local orientations.

First, process strengthened somewhat developers' cosmopolitan orientations. At higher CMM Levels, at least some staff spent more time reading industry journals and magazines and attending industry conferences:

In Program A, there was a focus on the big picture, and we tried to make development staff aware of other studies, findings, activities outside GCC. This sometimes prompted developers to become more interested in such conferences as the SEPG, DoD Tech conferences, and SEI affiliates symposia. (B process engineering, formerly with A)

The process-focused people certainly spend more time outside—the more you know, the more you know you don't know and need to learn. And probably some percentage of the general population spends more time outside as their awareness of things not-immediately-project increases and their interest is piqued. But my guess is that the John Q. project guy is still spending most of his time 'doing.' (D process engineering, also works with C)

At the same time, process maturity strengthened local orientations:

On the one hand, as higher levels of maturity are attained, many of the good developers realize their value and are now more marketable outside; on the other hand, they have 'bonded' with the organization and may be a bit more inclined to stay. (B process engineering, formerly with A)

Programs that don't have a process focus tend to have less of a formal connection with the larger GCC corporation. Programs focusing on advancing their process maturity tend to receive support from outside the immediate project, and tend to reach across GCC for best practices and lessons learned. The connection does create enhanced awareness, and in that sense perhaps both more 'professionalism' and more corporate identity. (D process engineering, also works with Program C)

A NEW PROFESSIONALISM

This more interdependent self implied a corresponding mutation in the nature of developers' notion of professionalism. Some aspects of professionalism were preserved, while some were significantly transformed.

On the one hand, process leveraged traditional values of professionalism, including the appeal to individual pride in the results of one's own work:

We appeal to people's sense of professionalism, saying something like, 'We're all professionals. And as professionals, we're both pretty mobile *and* committed to high quality work. Since I may leave here at some point, even soon, it's my duty as a professional to give the organization the documentation it needs to continue serving the customer.' (B quality assurance)

Our process makes for better testing, which means earlier detection of problems, which in turn makes the life of the programmer a lot easier and avoids a lot of embarrassment. (B department manager, formerly with A)

On the other hand, however, process seemed to encourage a mutation of professionalism. Whereas traditional conceptions of professionalism give great salience to the individual practitioner's judgement and thus to their autonomy—if not economic autonomy, at least technical decision-making autonomy—process encouraged the emergence of a more collective professional subject. This was expressed eloquently by this interviewee:

> Think of bridge building. Back in the eighteenth century, there were some very beautiful bridges built, but quite a few of them collapsed because they were designed by artists without any engineering understanding. Software is like bridge building. Software developers think of software as something of an art, and yes, you need that artistry, but you better have the engineering too. Developers often don't like the constraining rules, but the rules are necessary if you want to build complex things that have to work together. If you have only two or three people, you don't need all these rules. But if you have hundreds of people, the way we have here, then you need a lot of rules and discipline to get anything done. (C training)

This mutation is particularly significant because it appeared to moderate the traditional tension between professional autonomy and bureaucratic authority:

> Usually people run away from audits. But amazingly, recently we've seen several projects volunteering—they want to show off their accomplishments and process capabilities. (A process engineer)

> The Improvement Team's work ... made everyone realize that there are real business benefits to sharing information—instead of just worrying about your own rice bowl. I'm your [internal] customer, so I need you to understand my requirements. And the effect has been to make people interested in improving their own operations on their own, even without management being involved or pushing them. (D logistics)

TENSIONS: INTERDEPENDENCE VERSUS DEPENDENCE AND INDEPENDENCE

Interviews revealed two main types of tensions that could provoke resistance to the more socialized development process and thus affect the emergence of a more interdependent self.

First, due to the use-value/exchange-value contradictions at each of the other nodes, there was the constant risk that the demand for interdependence would mutate into coercive dependence and provoke either resistance or apathy. Second, developers sometimes clung to their independence. This was due in part to the tension between the collective and collaborative requirements of effective process (the use-value aspect) and the individual and competitive nature of the employment contract (the exchange-value aspect). Ultimately, the capitalist employment relation recognizes individuals, not collectivities. Even in firms with extensive team- and organization-wide rewards—and as noted above, GCC had few—these are minor components of the individual rank-and-file employee's compensation. The 'collective worker'—the community as productive actor—is in contradiction with the individualistic and instrumental foundation of the wage relation. In part it was also due to the contradictions between prior socialization and the demands of the new model for a new self-construal. In the US context, the change from a more independent to a more interdependent self-construal means a change in deeply ingrained cognitions and emotions. Such a change challenges the subject's prior socialization, as effected in early childhood personality formation, and in subsequent education, training, and work experiences.

Together, these contradictions help explain a certain lack of interest and passivity on the part of several interviewees:

> I follow the rules because they are there. (B developer)
>
> By and large, people just accept the Instructions pretty passively. (C development manager)

The overall result of these tendencies and counter-tendencies was an uneven process of subjective socialization. Expanding on the excerpt above:

> We still have to deal with the 'free spirits' who don't believe in process. These are typically people who have worked mainly in small teams. It's true that a small group working by itself doesn't need all this process. But we rarely work in truly independent small teams: almost all our work has to be integrated into larger systems, and will have to be maintained by people who didn't write the code themselves. These free spirits, though, are probably only between 2% and 4% of our staff. We find some of them in our advanced technology groups. We have some in the core of our business too, because they are real gurus in some complex technical area and we can't afford to lose them. And there are some among the new kids coming in too: many of them need convincing on this score. Most of them adapt, although some don't and they leave. (C process engineer)

Discussion

Several features mark the emerging form of community here as collaborative. In their values, structure, and identity, these organizations were constructing a form of community that was distinct from *Gemeinschaft* or *Gesellschaft*, and as distinct from the traditional, guildlike form of professionalism. Clearly, the salience of this form of community in structuring real working relations was rendered precarious by the pressures of valorization; but just as clearly, these organizations had made considerable progress towards the new form.

Values: interdependent contribution

The guildlike form of professional identity expressed in the hacker ideal of technical autonomy was progressively replaced under more process-mature conditions by an ethos of collaborative contribution—the software engineer as bridge builder. What mattered was one's contribution to the success of a complex organization rather than one's individual prowess—being part of a successful NBA team rather than a creative streetball player.

In the classic terms of cross-cultural analysis, we would say that the values of developers had become considerably more 'collectivist'; but there was clearly something distinctive about this collectivism. The new value orientation can be described with the two-dimensional mapping of culture and values articulated by Triandis and others: on one dimension, collectivism is contrasted with individualism, and on the other dimension, 'horizontal' cultures and low power distance are contrasted with 'vertical' cultures and high power distance.[49] Hackers, like other traditional professionals, are typically high on individualism and low on power distance. By contrast, while high-maturity software developers remain low on power distance and relatively egalitarian, they are more collectivist, a constellation we can call 'horizontal-collective.'[50]

Organization: interdependent process management

We saw how process maturity transformed the coordination of work: As the process became more mature and disciplined, the horizontal division of labor deepened; the span of integration became correspondingly broader; but at the same time, these relations became both tighter and more collaborative. Guild independence was replaced by complex interdependence. Guild independence was also undermined by the rationalization of an apprenticeship process that formalized the acquisition of firm-specific know-how.

Crucially, this interdependence did not take the form of hierarchical domination. The community principle prevailed over—but did not eliminate—the importance of hierarchy. These GCC programs appeared to be engaged in an effort to simultaneously maintain a high level of influence by lower-level developers *and* dramatically increase the level of organizational control.

Organizational research often proceeds as if influence by the rank-and-file professional and organizational control were mutually exclusive, in a zero-sum relation.[51] This case reminds us that they are not. In the four programs studied here, and in particular the two Level 5 ones, Program A and Program C, both higher-level and lower-level influence were relatively strong:

- Control was achieved through a panoply of mechanisms. Higher levels of the management hierarchy made overall architectural choices. Standardization of processes allowed operational decision making to be safely decentralized (per March and Simon's model of premiss setting). Management exercised strong control over project schedules and plans. And management actively structured mutual adjustment between people and units. The autonomy of individuals and units was thus severely curtailed. Overall coordination and consistency of action was not accidental or emergent, but planned through centralized control.
- Lower-level influence was also strong, as witnessed by the extensive involvement of staff in decision making and in developing and refining the standards, plans, and mutual adjustment processes under which they all worked. Management was committed to the philosophy of 'People support what they help create.' In the Level 5 programs, lower-level staff often participated in process tailoring; in contrast, such participation was significantly less common in the Level 3 programs. The Level 5 programs had 'improvement teams' in every subunit; in contrast, such teams were only occasionally operative in the Level 3 programs.

Identities: interdependent self-construals

The self is always social, always the result of a socialization process. It was just as social, just as much the result of individual socialization, when it took the hacker form. The independent self of the hacker was the result of socialization under a division of labor that was (using Durkheim's terms somewhat metaphorically) 'mechanical'—each person working in parallel on a self-contained task—rather than 'organic'—each person contributing only a specialized component of the whole. This independent self was reinforced by the shared norms of the profession, by educational experience, and by its fit with the relatively primitive tools, rules, and division of labor that then characterized the software development activity system. With greater process maturity and more advanced means of production, the social character of the self is no longer merely an abstract proposition, but becomes concrete in the form of subjectively experienced interdependent self-construals. The self is socialized—as it always has been—but now this socialization is not merely a remote antecedent, but becomes a lived reality.

In the present case, the experience of work under process maturity appeared to promote, first, a more interdependent sense of self: people's reference groups broadened. Second, process maturity led to a richer, more complex set of identities: we saw a heightened identification both with the broader software profession and with the local GCC corporation. Third, process promoted values that were, I argued, horizontal-collectivist, and corresponding to these values, a distinctive self-construal that preserves a certain individual autonomy even within growing interdependence.

Most cultural psychology contrasts *independent* and *interdependent* self-construals—but does not distinguish carefully between the latter and what we might more properly call *dependent* self-construals. This distinction seems crucial to differentiate strongly 'group'-based cultures and the form of professionalism we saw at GCC. The interdependent self-construals we heard expressed at GCC provide a less conformist form of other-directedness—a form of caring.[52] Viewed developmentally, one might characterize this orientation as representing a dialectical synthesis (transcendence) of the contradiction between *Gemeinschaft*, with its associated 'engulfment' of the individual,[53] and the modern, alienated autonomous individual under *Gesellschaft* conditions, a synthesis Hirschhorn (1997) calls the 'post-modern' self. We could call it the collaborative self.

Conclusion

The motivation for the present study was to understand the transformations of professional work through an analysis of the case of software development. I found that progress towards CMM Level 5 process maturity prompted the emergence of a new, collaborative form of professional community and of identity.

Marx contrasted the isolation of individuals and communities in the pre-capitalist world with the growing interdependence fostered by capitalist development. He saw capital's civilizing mission to abolish 'rural idiocy' and 'craft idiocy,' and to augment humanity's capacity for large-scale, productive collaboration. According to Marx, the development of capitalism is shaped by the basic contradiction between, on the one hand, the growing socialization of the forces of production and, on the other hand, the persistence of private ownership in the relations of production. Few today have Marx's confidence about the final outcome. But the present study suggests that this basic contradiction continues to shape the evolution of work, as illustrated here by the path traced by process maturity efforts—taking software development beyond 'hacker idiocy.'

Appendix 5.1: Four programs

For the main part, my analysis abstracts from the differences between the four programs. But the programs differed in many ways, as did the departments within them. Some background is therefore useful.

Program A: CMM Level 5. Program A had had a continuous contractual relationship with its customer for thirty years. Many employees had been attracted to the program because of the high public profile of the customer. Historically, Program A had ten to twenty projects under way at any one time, each building mid-sized subsystems composed of 100,000–400,000 lines of source-code. Recent years, however, had seen a downsizing of the organization due to the changes in the customer's needs. Between 1995 and 1999, employment had shrunk from over 1,600 to around 450.

Program A relied mainly on established technology and was responsible for a considerable amount of software maintenance. Over time, however, the program's tasks had become more complex as the customer requirements and the associated technologies had evolved. The business environment had also become more demanding, with considerable pressure for more code reuse and tighter deadlines.

Discussing the history of process at Program A, one interviewee summarized the evolution in these terms:

> 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)

Program A adopted the CMM in the early 1990s, and during the latter part of the decade used both CMM and ISO-9001 to help guide their improvement efforts. While their customer did not require CMM certification, external pressure played an important role in its adoption: 'We knew that other clients would require it and we felt it might be a good thing to do to help us improve' (B process engineering, formerly with A).

In 1991 the first formal, external Software Capability Evaluation (SCE) rated the organization a Level 1. The organization subsequently undertook several internal self-assessments. In 1996, the second evaluation rated them close to Level 3. In 1998, they were formally assessed as a Level 5 organization.

Program A was a 'poster child' for aggressive process improvement efforts at GCC. Analyzing some thirty projects over the 1994–2000 timeframe, Program A found that both productivity and quality improved on average by 10 per cent per year. They also saw dramatic improvement in the accuracy and speed with which they forecast costs and schedules for project proposals.

An internal survey of Program A personnel in 1999 asked whether they saw value in the effort associated with CMM certification. There were 260 responses from 850 surveys distributed. Opinions were largely positive, and more so among people who had personally participated in an audit. Of those not audited for the CMM, 58 per cent saw CMM as 'well worth the effort' and another 30 per cent as 'of some value.' Of those audited, 79 per cent thought CMM was 'well worth the effort' and another 18 per cent thought it was 'of some value.' The proportion who thought it was of little or no value was 12 per cent among the non-audited and 3 per cent among the audited.

Program B: CMM Level 3. Program B's mission was to build information management tools for its government client to use in operations around the world: internal accounting, management, IS resource management, and so on. Program B's staff developed new systems, maintained and upgraded old ones, and operated 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 eighteen months. GCC replaced nearly thirty contractor organizations that had worked largely independently of each other. Program B functioned as a prime contractor and system integrator, both developing systems itself and coordinating a small number of subcontractors.

Program B itself employed directly or indirectly about 275 people. The largest of its projects employed about ninety people. The overall system was composed of 700 files, comprising about 1 million lines of source-code (MSLOC).

To help reach Level 3, several people were transferred from Program A. The two largest projects were each led by former Program A people, and Program A veterans staffed several other key management and staff positions. Program B's process efforts were slowed down by a very difficult Y2K project, which strained relations with the client. Once that project was completed, relations improved. The program was officially assessed as Level 3 in early 2000.

Program C: CMM Level 5. This program, like Program A, had had a continuous contractual relationship with its Department of Defense (DoD) end-customer for some thirty years. But the relationship had always been mediated by other organizations serving as prime contractors. Program C undertook two or three major projects at a time, each representing about 2.5 MSLOC. These projects created new versions of the weapons system control software they provided to the DoD. Program C employed about 450 people. It was divided into four main units that developed and maintained the main modules of the system, plus several support departments.

In recent years, there had been a swing towards greater bottom-up participation and a corresponding effort to change management styles.

I think it's fair to say that our culture has not put a lot of weight on things like participation and empowerment. When I first came on board, I found levels of secrecy, need to know, and cones of silence. In part, that was due to the influence of the customer we worked with, and to the high proportion of senior people both there and here with military backgrounds. It was like, 'Don't ask questions; just do it,' and the ethos around here was more like 'Just do your job.' Even the TQM program in the early 1990s didn't make much of an impact. It was seen mainly as a passing fad. But in the last 18 months, the change has been dramatic. We've started to free up resources for symbolic and financial recognition. And we've emphasized communication more. (C process engineering)

The key drivers of process maturity at Program C had historically 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) (see Schulmeyer 1998 for an overview of the evolution of the DoD software standards). Unlike Programs A and B, Programs C and D did not have dedicated process engineering groups driving process improvement, but relied instead on an expanded quality assurance staff and cross-functional management-level Software Engineering Process Groups (SEPGs). Nevertheless, 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. Shortly thereafter, it was evaluated at Level 5. The quality of its products was widely recognized. The program had averaged 97 per cent of award fees, which is an unusually high rate among DoD contractors.

Program D: CMM Level 3. Program D began operations in 1991, developing infrastructure systems for the DoD. Program D was unusual within GCC because it covered the whole product life/cycle, offering complete solutions including hardware, the integration of hardware and software, warehousing, installation, and ongoing support. It had developed 2 MSLOC over the 1993–9 period. Program D was also unusual within GCC for its extensive use of commercial, off-the-shelf (COTS) hardware and software. Its systems incorporated over 200 commercial products. The program's systems were being used in about 100 sites, of which about 50 were interlinked. In 2000, Program D employed some 350 people directly, plus a further 120 contractors.

Until recently, software process had received less attention in this program than in the others studied. According to one interviewee:

> The Program D system was billed as based on a prototyping approach rather than the traditional waterfall approach. At the time, this was pretty leading-edge stuff at GCC, and it attracted people who don't like the discipline associated with relatively routine projects of the kind GCC Government Systems had traditionally done. But the initial team of 30 people has grown to nearly 600, and the business really has to deliver, so they realized they needed at least some Level 2 discipline. Even some of the die-hards have had a kind of religious conversion, and have become quite committed to process now. (D process engineering, also works with Program C)

Process had recently moved into the foreground. 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, they conducted their own assessment, and discovered that the program would likely be rated no higher than a Level 1. 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 1999, the program was assessed as Level 3.

Differences across the programs. Contingency theory teaches us that the scope for process standardization and formalization such as recommended by the CMM is closely related to the degree of routineness of the organization's key tasks. Task routineness varied across these GCC programs — that is, people encountered more or fewer exceptions to established patterns of problem solving, and these exceptions were more or less difficult to resolve. And as predicted by contingency theory, the level of detail in the process varied across programs with this routineness. For example, in comparison with Program C, its sister program D dealt with a broader range of technologies and these technologies evolved more rapidly; so it is not surprising that 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 and degree of process discipline. 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 development, did specify Instructions.

Appendix 5.2: Cultural-historical activity theory

This paper relies on cultural-historical activity theory (CHAT) and the broader Marxist theory on which CHAT is based. As developed primarily by Yjro Engeström and Michael Cole, CHAT is based largely on the work of Vygotsky, Luria, and Leont'ev.[54] CHAT is distinctive first in its unit of analysis, positing that the most appropriate unit of analysis for the study of work situations is the *activity*, understood as the system of relations established when a community pursues a common object. Leont'ev argues that in the study of work, the activity is the 'molar' unit of analysis, preferred over discrete psychological *operations* (quasi-automatic responses to stimuli) because work involves higher-order cognitive functions, and preferred over *actions* (discrete goal-directed behaviors) because work is best understood as a collective endeavor that unfolds over historical time.[55]

Adapting Engeström's development of CHAT, and summarized in Figure 5.1, the structure of an activity system can be understood as an interrelated set of functional nodes:

- Subject: The subject of activity can be an individual or a collective actor.
- Object: The object of an activity is not merely the behaviorist's stimulus, but (consistent with common usage, American Pragmatism, and Marx of the 1844 Manuscripts) both (*a*) something given to the mind or senses, and (*b*) a purpose. The gap between these two is the fundamental 'motivation' for the activity, and the object thus has a profound effect on the shape of the rest of the activity system.
- Tools: Human activity is distinctive in its extensive reliance on tools (including concepts and language) as culturally transmitted artifacts that partly mediate the subject's relation to the object of activity.
- Community: The subject's activity is embedded in the activity of community that shares the same object. At the community node, CHAT captures the contours of the subject's reference group, its internal division of labor, and the rules governing the behavior of individuals in the community.[56]
- Outcome: The result of the system's activity is an outcome, a product. This result will, in some cases, become an input to another activity system. Conversely, nodes of the focal system can themselves be the outcome of other activity systems: tools, for example, are often procured from specialized suppliers.
- More advanced model: In some circumstances, actors in the activity system are aware of a potentially more advanced model of their activity, and the contradiction between this ideal and the current form of the activity system—along with the other contradictions discussed below—can drive the evolution of the system.

Figure 5.1 is a model of what Marx calls 'production in general'—the generic structure of productive activity; as it stands, this is a trans-historical model that abstracts from the specific forms that activity takes in different social settings.[57] To understand any given activity, we need to augment this model with the more concrete determinations that reflect the nature of the specific cultural-historical context of activity. CHAT analyzes the resulting model in terms of its characteristic contradictions. Following Engeström, we can distinguish four types of contradiction, and in the analysis presented in the body of the chapter, we will see examples of each:

- Each node is marked by a primary contradiction characteristic of all activities (but particularly business activities) conducted in capitalist social settings—the contradiction between use-value and exchange-value. I find that use-value considerations in the implementation of the CMM drive the software development activity system towards greater interdependence and greater efforts to master that interdependence through more collaborative community. Exchange-value considerations, on the other hand, sometimes stimulate these forces and sometimes impede them as unprofitable, replacing collaboration with competition and autocratic control.
- Secondary contradictions give rise to tensions between nodes. In the present study, I focus mainly on the contradictions that link the community and subject nodes to the others. These are the proximate cause of the emergence of a new form of professional community and new self-construals among developers.
- Tertiary contradictions are those between the form of the current activity system and a more advanced model of it. I find that the CMM functions in a manner akin to what activity-oriented psychologists have called 'scaffolding,'[58] guiding an organizational learning process that also contributes directly and indirectly to the emergence of new forms of community and identity.
- Quaternary contradictions are those between the given activity system and surrounding activity systems. On the outcome side, I find contradictions between the interests of development organizations and those of their clients. On the input side, I find contradictions between the prior socialization of developers and the subjective demands of more mature development processes.

CHAT offers a particularly fruitful path for understanding the mutations of software development by interpreting change as resulting from these various contradictions and their interactions.

References

Abbott, A. (1988). *The System of Professions: An Essay on the Division of Expert Labor.* Chicago: University of Chicago Press.

Adler, P. S. (1996). 'The dynamic relationship between tacit and codified knowledge: comments on Nonaka's ''Managing innovation as a knowledge creation process.'' ' In G. Pogorel and J. Allouche (eds.), *International Handbook of Technology Management*. Amsterdam: North-Holland: 110–24.

—— (1999). 'Building better bureaucracies.' *Academy of Management Executive*, 13/4: 36–47.

—— (2001). 'Market, hierarchy, and trust: the knowledge economy and the future of capitalism.' *Organization Science*, 12/2: 214–34.

—— and Borys, B. (1996). 'Two types of bureaucracy: enabling and coercive.' *Administrative Science Quarterly*, 41/1: 61–89.

—— Kwon, S., and Singer, J. (2003). 'The ''Six West'' problem: an essay on the role of professionals in knowledge management, with particular reference to the case of hospitals.' USC working paper.

Azjen, I. (1991). 'The theory of planned behavior.' *Organizational Behavior and Human Decision Processes*, 50: 179–211.

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

—— (1995). 'Enough about process: what we need are heroes.' *IEEE Software*, 12/2: 96–8.

Bakhurst, D., and Sypnowich, C. (1995). 'Introduction: problems of the social self.' In D. Bakhurst and C. Sypnowich (eds.), *The Social Self*. London: Sage: 1–17.

Bandura, A. (1997). *Self-Efficacy: The Exercise of Control*. New York: W. H. Freeman.

Barley, S. R. (1986). 'Technology as an occasion for structuring: evidence from observations of CT scanners and the social order of radiology departments.' *Administrative Science Quarterly*, 31/1: 78–108.

—— and Tolbert, P. S. (1997). 'Institutionalization and structuration: studying the links between action and institution.' *Organization Studies*, 18/1: 93–117.

Baronas, A., and Louis, M. (1988). 'Restoring a sense of control during implementation: how user involvement leads to system acceptance.' *MIS Quarterly*, 12/1: 111–24.

Bart, C. K. (1999). 'Controlling new products: a contingency approach.' *International Journal of Technology Management*, 18/5–8: 395–413.

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.

Blau, P. M. (1955). *The Dynamics of Bureaucracy*. Chicago: University of Chicago Press.

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, Mass.: Harvard University Press.

Braverman, H. (1974). *Labor and Monopoly Capital*. New York: Monthly Review Press.

Brown, S. L., and Eisenhardt, K. M. (1995). 'Product development: past research, present findings, and future directions.' *Academy of Management Review*, 20/2: 343–78.

Burkitt, I. (1991). *Social Selves: Theories of the Social Formation of Personality*. London: Sage.

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 (July): 374–85.

Butler, T., Standley, V., and Sullivan, E. (2001). 'Software configuration management: a discipline with added value.' *Crosstalk*, 14/7: 4–8.

Chaiklin, S., and Lave, J. (eds.) (1993). *Understanding Practice: Perspectives and Activity and Context*. New York: Cambridge University Press.

—— 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.

Cohen, G. A. (1974). 'Marx's dialectic of labor.' *Philosophy and Public Affairs*, 3/3: 235–61.

—— (1978). *Karl Marx's Theory of History: A Defense*. Princeton: Princeton University Press.

Cohen, Joshua (1982). Review of G. A. Cohen, *Karl Marx's Theory of History. Journal of Philosophy*, 79/5: 253–73.

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

Conn, R. (2002). 'Developing software engineers at the C-130J software factory.' *IEEE Software* (Sept.–Oct.): 25–9.

Conradi, R., and Fuggetta, A. (2002). 'Improving software process improvement.' *IEEE Software* (July–Aug.): 92–9.

Cooper, D. J., Hinings, C. R., Greenwood, R., and Brown, J. L. (1996). 'Sedimentation and transformation in organizational change: the case of Canadian law firms.' *Organization Studies*, 17/4: 623–47.

Couger, J. D., and O'Callaghan, R. (1994). 'Comparing the motivation of Spanish and Finnish computer personnel with those of the United States.' *European Journal of Information Systems*, 3/4: 285–301.

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.

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

—— (2000). ' ''Made in India'': a new sign of software quality.' *Computerworld* (29 Feb.): 36.

Damanpour, Fariborz (1991). 'Organizational innovation.' *Academy of Management Journal*, 34: 555–91.

Dannefer, D. (1984). 'Adult development and social theory: a paradigmatic reappraisal.' *American Sociological Review*, 49: 100–16.

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

Derber, C. (1982). *Professionals as Workers: Mental Labor in Advanced Capitalism*. Boston: G. K. Hall & Co.

Dewey, J. (1930). *Individualism Old and New*. New York: Minton, Balch & Co.

—— (1939). 'The individual in the new society.' In J. Ratner (ed.), *Intelligence in the Modern World: John Dewey's Philosophy*. New York: Random House: 405–33.

Dosi, G. (1996). 'The contribution of economic theory to the understanding of a knowledge-based economy.' In *Employment and Growth in the Knowledge-Based Economy*. Paris: Organization of Economic Cooperation and Development: 81–94.

Eisenhardt, K. M., and Tabrizzi, B. N. (1995). 'Accelerating adaptive processes: product innovation in the global computer industry.' *Administrative Science Quarterly*, 40/1: 84–111.

Elias. N. (1998). *On Civilization, Power, and Knowledge: Selected Writings*, ed. S. Mennell and J. Goudsblou. Chicago: University of Chicago Press.

—— (2000). *The Civilizing Process*. Malden, Mass.: Blackwell.

Engels, F. (1978). 'Socialism: utopian and scientific.' In R. C. Tucker (ed.), *The Marx-Engels Reader*. 2nd edn. New York: Norton: 683–717.

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

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

—— (1999). 'Situated learning at the threshold of the new millennium.' In J. Bliss, R. Säljö, and P. Light (eds.), *Learning Sites: Social and Technological Resources for Learning*. Amsterdam: Pergamon.

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

Erez, M., and Earley, P. C. (1993). *Culture, Self-Identity, and Work*. New York: Oxford University Press.

Fiske, A. P., Kitayama, S., Markus, H. R., and Nisbett, R. E. (1998). 'The cultural matrix of social psychology.' In D. T. Gilbert, S. T. Fiske, and G. Lindzey (eds.), *The Handbook of Social Psychology*. 4th edn. Boston: McGraw-Hill: 915–81.

Freidson, E. (2001). *Professionalism: The Third Logic*. Cambridge: Polity.

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

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

Gibbs, G. G. (1994). 'Software's chronic crisis.' *Scientific American* (Sept.): 86–92.

Gibson, C. D., and Earley, P. C. (n.d.). 'Work-team performance motivated by collective thought: the structure and function of group efficacy.' Unpublished, University of Southern California.

Giddens, A. (1979). *Central Problems in Social Theory*. Berkeley and Los Angeles: University of California Press, Goldstein, D. K., and Rockart, J. F. (1984). 'An examination of work-related correlates of job satisfaction in programmer/analysts.' *MIS Quarterly*, 8/2: 103–15.

Gordon, R. W., and Simon, W. H. (1992). 'The redemption of professionalism?' In R. L. Nelson, D. M. Trubek, and R. L. Solomon (eds.), *Lawyers' Ideals/Lawyers' Practice: Transformations in the American Legal Profession*. Ithaca, NY: Cornell University Press.

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

—— (1957). 'Cosmopolitans and locals: toward an analysis of latent social roles.' *Administrative Science Quarterly*, 2/3: 281–306.

Gramsci, A. (1971). *Selections from the Prison Notebooks*. New York: International Publishers.

Green, G., and Hevner, A. R. (1999). 'Perceived control of software developers and its impact on the successful diffusion of information technology.' Carnegie Mellon University, Software Engineering Institute, Special Report CMU/SEI-98-SR-013.

Greenbaum, J. M. (1979). *In the Name of Efficiency*. Philadelphia: 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 23. San Francisco: Jossey-Bass: 45–63.

Griss, M. L. (1993). 'Software reuse: from library to factory.' *IBM Systems Journal*, 32/4: 548–66.

Hackman, J. R., and Oldham, G. R. (1980). *Work Redesign*. Reading, Mass.: Addison-Wesley.

Harter, D. E., Krishnan, M. S., Slaughter, S. A. (2000). 'Effects of process maturity on quality, cycle time, and effort in software development.' *Management Science*, 46/4: 451–66.

Henderson, J., and Lee, S. (1992). 'Managing I/S design teams: a control theories perspective.' *Management Science*, 38/6: 757–76.

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.

Hernandez, M., and Iyengar, S. S. (2001). 'What drives whom? A cultural perspective on human agency.' *Social Cognition*, 19/3: 269–94.

Hirschhorn, L. (1997). *Reworking Authority*. Cambridge, Mass.: MIT Press.

Hoch, D. J., Roeding, C. R., Purkert, G., and Linder, S. K., (2000). *Secrets of Software Success*. Boston: Harvard Business School Press.

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.

Hyman, R. (1987). 'Strategy or structure? Capital, labour and control.' *Work, Employment and Society*, 1/1: 25–55.

Jackson, S. E., and Schuler, R. S. (1985). 'A meta-analysis and conceptual critique of research on role ambiguity and role conflict in work settings.' *Organizational Behavior and Human Decision Processes*, 36: 17–78.

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

Jones, C. (2002). 'Defense software development in evolution.' *CrossTalk* (Nov.): 26–9.

Kagitcibasi, C. (1997). 'Individualism and collectivism.' In J. W. Berry, M. H. Segall, and C. Kagitcibasi (eds.), *Handbook of Cross-Cultural Psychology*. Needham Heights, Mass.: Allyn & Bacon: 1–49.

Kahn, R., Wolfe, D., Quinn, R., Snoek, J. D., and Rosenthal, R. (1964). *Organizational Stress: Studies in Role Conflict and Role Ambiguity*. New York: John Wiley and Sons.

Kenney, M., and Florida, R. (1993). *Beyond Mass Production: The Japanese System and its Transfer to the U.S.* New York: Oxford University Press.

King, R. C., and Sethi, V. (1998). 'The impact of socialization on the role adjustment of information system professionals.' *Journal of Management Information Systems*, 14/1: 195–217.

Kogut, Bruce, and Metiu, Anca (2001). 'Open-source software development and distributed innovation.' *Oxford Review of Economic Policy*, 17: 248–64.

Kohn, M. L., and Schooler, C. (1983). *Work and Personality*. Norwood, NJ: Ablex.

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.

Kunda, G. (1992). *Engineering Culture: Control and Commitment in a High-Tech Corporation*. Philadelphia: Temple University Press.

Langer, E. (1983). *The Psychology of Control*. Beverly Hills, Calif.: Sage.

Lave, J. (1988). *Cognition in Practice*. New York: Cambridge University Press.

—— (1993). 'The practice of learning.' In S. Chaiklin and J. Lave (eds.), *Understanding Practice: Perspectives and Activity and Context*. New York: Cambridge University Press: 3–35.

—— (2001). 'Lines on social practice theory.' Unpublished manuscript, UC Berkeley.

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

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

Levine, A., and Wright, E. (1980). 'Rationality and class struggle.' *New Left Review*, 123: 47–68.

Lieberman, H., and Fry, C. (2001). 'Will software ever work?' *Communications of the ACM*, 44/3: 122–4.

Lillkrank, P. (2003). 'The quality of standard, routine and nonroutine processes.' *Organization Studies*, 24/2: 215–33.

Livingston, J. (2000). 'The strange career of the "social self." ' *Radical History Review*, 76: 53–79.

Luria, A. R. (1976). *Cognitive Development: Its Cultural and Social Foundations*. Cambridge, Mass.: Harvard University Press.

Lynn, L. H. (1991). 'Assembly line software development.' *Sloan Management Review*, 88:

McKinlay, J. B., and Arches, J. (1985). 'Toward the proletarianization of physicians.' *International Journal of Health Services*, 15/2: 161–95.

March, J., and Simon, H. (1958). *Organizations*. New York: Wiley.

Markus, H. R., and Kitayama, S. (1991). 'Culture and the self: implications for cognition, emotion, and motivation.' *Psychological Review*, 98: 224–53.

Marx, K. (1955). *The Poverty of Philosophy*. Moscow: Progress. Online version consulted 27 Dec. 2004 at www.marxists.org/archive/marx/works/1847/poverty-philosophy/ch026.htm.

—— (1973). *Grundrisse*. Harmondsworth: Penguin Books.

—— (1975). *Karl Marx: Early Writings*. New York: Vintage.

—— (1977). *Capital*. Vol. i. New York: Vintage.

—— and Engels, F. (1959). 'The Communist Manifesto.' In L. S. Feuer (ed.), *Marx and Engels: Basic Writings on Politics and Philosophy*. New York: Anchor: 1–41.

Mathieson, K. (1991). 'Predicting user intentions: comparing the technology acceptance model with the theory of planned behavior.' *Information Systems Research*, 2/3: 173–91.

Meyer, J. W., and Rowan, B. (1977). 'Institutionalized organizations: formal structure as myth and ceremony.' *American Journal of Sociology*, 83: 340–63.

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

Mowery, D. C. (ed.) (1996). *The International Computer Software Industry*. New York: Oxford University Press.

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*. Cambridge, Mass.: MIT Press: 69–102.

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.

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.

Oyserman, D., Coon, H. M., and Kemmelmeier, M. (2002). 'Rethinking individualism and collectivism: evaluation of theoretical assumptions and meta-analyses.' *Psychological Bulletin*, 126/1: 3–73.

Paulk, M. C. (1995). 'The evolution of SEI's Capability Maturity Model for Software.' *Software Process: Improvement and Practice*, 1: 3–15.

—— Weber, Charles V., Garcia, Sozanne M., Chrissis, Mary Beth, and Bush, Marilyn W. (1993). 'Key practices of the capability maturity model, version 1.1.' Software Engineering Institute, CMU/SEI-93-TR-25, DTIC No. ADA263432, Feb.

Pinnington, A., and Morris, T. (2003). 'Archetype change in professional organizations: survey evidence from large law firms.' *British Journal of Management*, 14/1: 85–99.

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*, 29/4: 820–31.

Rasch, R. H., and Tosi, H. L. (1992). 'Factors affecting software developers' performance: an integrated approach.' *MIS Quarterly* (Sept.): 395–413.

Rizzo, J. R., House, R. J., and Lirtzman, S. I. (1970). 'Role conflict and ambiguity in complex organizations.' *Administrative Science Quarterly*, 15: 150–63.

Sacks, M. (1994). *On-the-Job Learning in the Software Industry*. Westport, Conn.: Quorum.

Sagie, A. (1997). 'Leader direction and employee participation in decision-making: contradictory or compatible practices?' *Applied Psychology: An International Review*, 46: 387–416.

Schneider, B. (1987). 'The people make the place.' *Personnel Psychology*, 40: 437–54.

Schulmeyer, G. G. (1998). 'Standardization of software quality assurance: where is it all going?' In G. G. Schulmeyer and J. I. McManus (eds.), *Handbook of Software Quality Assurance*. Upper Saddle River, NJ: Prentice Hall: 61–90.

Smith, P. G., and Reinertsen, D. G. (1991). *Developing Products in Half the Time*. New York: Van Nostrand.

Software Engineering Institute (2002). 'Process maturity profile of the software community, 2002 mid-year update.' Download from www.sei.cmu.edu.

—— (2004). 'Process maturity profile software CMM, 2004 mid-year.' Download from www.sei.cmu.edu.

Sohn-Rethel, A. (1978). *Intellectual and Manual Labour: A Critique of Epistemology*. Atlantic Highlands, NJ: Humanities Press.

Spenner, K. I. (1990). 'Skill: meanings, methods, measures.' *Work and Occupations*, 17: 399–421.

Standish Group (1994). Chaos study report at www.standishgroup.com.

Steinmuller, W. E. (1996). 'The U.S. software industry: An analysis and interpretive history.' In D. C. Mowery (ed.), *The International Computer Software Industry*. New York: Oxford University Press: 15–52.

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*. New York: Oxford University Press: 169–83.

Strauss, A. L., Fagerhaugh, S., Suczek, B., and Wiener, C. (1985). *Social Organization of Medical Work*. Chicago: University of Chicago Press.

Suchman, L. (1987). *Plans and Situated Actions*. Cambridge: Cambridge 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* (Dec.): 567–79.

Tannenbaum, A. S. (ed.) (1968). *Control in Organizations*. New York: McGraw-Hill.

Taylor, C. (1989). *The Sources of the Self*. Cambridge: Cambridge University Press.

Taylor, S., and Todd, P. (1995). 'Understanding information technology usage: a test of competing models.' *Information Systems Research*, 6/2: 144–76.

Thompson, P. (1989). *The Nature of Work: An Introduction to Debates on the Labour Process*. London: Macmillan.

Triandis, H. C., and Gelfand, M. J. (1998). 'Converging measurement of horizontal and vertical individualism and collectivism.' *Journal of Personality and Social Psychology*, 74/1: 118–28.

—— and Suh, E. M. (2002). 'Cultural influences on personality.' *Annual Review of Psychology*, 53: 133–60.

—— Leung, K., Villareal, M., and Clack, F. L. (1985). 'Allocentric versus idiocentric tendencies: convergent and discriminant validation.' *Journal of Personality Psychology*, 19: 395–415.

van der Pijl, K. (1998). *Transnational Classes and International Relations*. London: Routledge.

Van Iterson, A., Mastenbroek, W., Newton, T., and Smith, D. (eds.) (2002). *The Civilized Organization: Norbert Elias and the Future of Organization Studies*. Philadelphia: John Benjamins.

Van Maanen, J., and Barley, S. R. (1984). 'Occupational communities: culture and control in organizations.' *Research in Organizational Behavior*, 6: 287–365.

Vygotsky, L. S. (1962). *Thought and Language*. Cambridge, Mass.: MIT Press.

—— (1978). *Mind in Society*. Cambridge, Mass.: 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. Sharp.

—— Tulviste, P., and Hagstrom, F. (1993). 'A sociocultural approach to agency.' In E. A. Forman, N. Minick, and C. A. Stone (eds.), *Contexts for Learning: Sociocultural Dynamics in Children's Development*. New York: Oxford University Press: 136–56.

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

Wood, D., Bruner, J., and Ross, G. (1976). 'The role of tutoring in problem-solving.' *Journal of Child Psychiatry and Psychology,* 17: 89–100.

Wright, E. O., Levine, A., and Sober, E. (1992). *Reconstructing Marxism: Essays on Explanation and the Theory of History.* London: Verso.

Yates, J., and Orlikowski, W. J. (1992). 'Genres of organizational communication: a structurational approach to studying communication and media.' *Academy of Management Review,* 17/2: 299–323.

Notes

1. Gordon and Simon (1992: 234).
2. The literature on professionals and the mutations of professional work is enormous. An excellent entry point is provided by Eliot Freidson (2001).
3. e.g. McKinlay and Arches (1985); Derber (1982).
4. Greenbaum (1979: 64–5).
5. Gibbs (1994); Lieberman and Fry (2001); Standish (1994).
6. Harter et al. (2000); Clark (1999); Cusumano (2000).
7. On the concept of the software factory and the associated debates, see Cusumano (1991); Swanson et al. (1991); Griss (1993): Weber (1997); Friedman and Cornford (1989).
8. The general argument for this position is articulated in the 'contingency theory' branch of organization studies—see Burns and Stalker (1961); Lawrence and Lorsch (1967); Galbraith (1977); Mintzberg (1979). The argument has been applied in the software development arena by authors such as Crocca (1992); Bach (1994, 1995); Conradi and Fuggetta (2002); Lynn (1991); Ngwenyama and Nielson (2003).
9. Marx (1973, 1977); Engels (1978).
10. Markus and Kitayama (1991).
11. Marx, following Hegel, takes contradictions as real—out there, in objective, independent 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.
12. Marx (1977: appendix); Thompson (1989); Bottomore (1983: 267–70).
13. Engels (1978). My reading of Marx is based on G. A. Cohen's (1978) presentation. Cohen's version has been criticized by, amongst others, Levine and Wright (1980) and Cohen (1982); see G. A. Cohen's (1988) reply, also Wright et al. (1992). This essay takes G. A. Cohen's interpretation from the general societal plane into the production process.
14. e.g. Marx (1973: 705; 1977: 1024).

15. See also van der Pijl (1998); Engels (1978).
16. Engels (1978: 702).
17. Marx (1977: ch. 13).
18. Marx (1977: 458); Gramsci (1971: 201–2).
19. Markus and Kitayama (1991).
20. Marx (1977: 447).
21. Marx and Engels (1959); Marx (1955).
22. Marx (1973: 704–6).
23. Hyman (1987).
24. Adler (2001).
25. See Cohen (1978); Levine and Wright (1980).
26. Humphrey (2002).
27. Jones (2002).
28. Paulk (1995: 11).
29. Crosby (1979); see Humphrey (2002); a bibliography on the CMM is available at www.sei.cmu.edu/docs/biblio.pdf.
30. Clark (1999).
31. Harter et al. (2000). Other multi-organization studies include Krishnan et al. (2000); Herbsleb et al. (1997).
32. For some observers, the bureaucratizing path of the CMM is a dead end, and the way forward is shown by the open-source movement and 'agile' methods. But close examination shows that these approaches are only feasible in narrow segments of the software industry. Open source is feasible only where interfaces can be standardized and systems can be modularized (Kogut and Metiu 2001); and its cost-effectiveness remains unproven. 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—see Boehm and Turner (2003).
33. Autonomy is often presented as a key motivating characteristic of jobs (Hackman and Oldham 1980). A similar assumption underlies Ajzen's (1991) theory of planned behavior, with its focus on 'perceived behavioral control.' In labor process theory and the broader field of sociology of work too, autonomy is one of the two defining dimensions of skilled work, alongside complexity (see Braverman 1974; Spenner 1990). In the Information Systems field, a considerable body of research has focused on the role of perceived control and autonomy as determinants of the use of, and satisfaction with, new techniques and technologies (see for example Baronas and Louis 1988; Mathieson 1991; Taylor and Todd 1995; Henderson and Lee 1992; Green and Hevner 1999). This emphasis on autonomy seems particularly appropriate for programmers, who typically manifest low need for social interaction (Couger and O'Callaghan 1994).
34. On the software industry and its components and major players, see Hoch et al. (2000), and Mowery (1996). Notwithstanding the growth of the personal computer market and the associated mass-market pre-packaged 'software prod-

ucts' industry, the bulk of the rapidly expanding software industry resembles GCC and its competitors in delivering 'software services'—creating fully or partly customized, large-scale systems for specific clients. In 2000, according to data provided by IDC, software services in the USA accounted for revenues of $395 billion versus $171 billion for software products. Steinmuller (1996) notes that both these industry segments are small relative to the software developed for their own use by firms and public organizations.

35. See Abbott (1988).
36. Kraft (1977: 56).
37. Adler (2001).
38. As described by Meyer and Rowan (1977).
39. Using Marx's periodization (1977: pt. IV).
40. For discussion of the impact of process maturity on the role of another key staff function, Configuration Management, see Butler et al. (2001): many of the same conclusions apply.
41. See for example Sacks (1994); Lave (1988).
42. The metaphor of scaffolding 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.' The metaphor was originally articulated by Wood et al. (1976). The concept of 'zone of proximal development' is one of Vygotsky's best-known contributions: see Griffin and Cole (1984).
43. 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 precisely the 'proleptic' manner described by Cole (1996: 183 ff.).
44. This concern echoed critiques of the 'top-down' nature of the scaffolding metaphor: see Stone (1993); Butler (1998).
45. See Wheelwright and Clark (1992); Smith and Reinertsen (1991).
46. Bandura (1997).
47. On adult socialization, see e.g. Kohn and Schooler (1983); and see Conn (2002) for discussion of the process of developer socialization in another software factory. On 'attraction-selection-attrition,' see Schneider (1987).
48. Gouldner (1957); Van Maanen and Barley (1984).
49. See Triandis and Gelfand (1998).
50. An alternative—more provocative but also more confusing—formulation would be to argue that individualism and collectivism are orthogonal constructs and the new developers are high on both: see discussion in Oyserman et al. (2002: 8), and other contributors to that issue; also Kagitcibasi (1997).
51. This shallow view has persisted notwithstanding the strong case made by Tannenbaum's work on the 'control graph'—see, e.g., Tannenbaum (1968); more recently, Henderson and Lee (1992).
52. On the social-self thesis and other-directedness, see Livingston (2000).
53. See Cohen (1974).

54. Engeström (1987, 1990); M. Cole (1996); Vygotsky (1962, 1978); Luria (1976); Leont'ev (1978). Useful overviews of CHAT and discussion of its variants include: Engeström et al. (1999); Chaiklin et al. (1999); Wertsch (1979); Blackler (1993); Holt and Morris (1993).
55. Leont'ev (1978).
56. It is here that my presentation differs from Engeström: Engeström distinguishes rules and division of labor from community itself, whereas I have subsumed them under the one broader category. This eases the burden of exposition.
57. Marx (1973: 85).
58. Wood et al. (1976).