Information and Organization 19 (2009) 277–293

Contents lists available at ScienceDirect

Information and Organization

journal homepage: www.elsevier.com/locate/infoandorg

Time as symbolic currency in knowledge work

Dariusz Jemielniak *

Kozminski University, Ul. Jagiellonska 59, 03-301 Warszawa, Poland

ARTICLE INFO

Article history:
Received 5 April 2008
Received in revised form 16 August 2009
Accepted 24 August 2009

Keywords:
Knowledge-intensive work
Time management
Software engineers' culture
Professions
High-tech
Schedules
Software delays

ABSTRACT

The paper discusses the issue of time slips in software development. Increasing time sacrifices toward work constitutes an important part of modern organizational environment. In fact, the reign over time is a crucial element in controlling the labor process. Yet a lack of cultural studies covering different approaches to this issue remains—particularly those focusing on high-skilled salaried workers.

This article is a small attempt to fill this gap, based on an analysis of unstructured qualitative interviews with high-tech professionals from a B2B software company. It focuses on the issue of timing in IT projects, as perceived by software engineers. The findings indicate that managerial interruptions in work play an important part in the social construction of delays. However, interruptions from peer software engineers are not perceived as disruptive. This leads to the conclusion that time is used in a symbolic way, both for organizational domination and solidarity rituals. The use of time as a symbolic currency in knowledge-work rites is presented as often influencing the very process of labor and schedules. It is revealed to be the dominant evaluation factor, replacing the officially used measures, such as efficiency, or quality.

A manager asked a programmer how long it would take him to finish the program on which he was working. "I will be finished tomorrow," the programmer promptly replied. "I think you are being unrealistic," said the manager, "Truthfully, how long will it take?" The programmer thought for a moment. "I have some features that I wish to add. This will take at least two weeks," he finally said.
"Even that is too much to expect, "insisted the manager, "I will be satisfied if you simply tell me when the program is complete." The programmer agreed to this.

* Tel.: +48 604901352.
E-mail address: darekj@wspiz.edu.pl

1471-7727/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.infoandorg.2009.08.002

Several years later, the manager retired. On the way to his retirement lunch, he discovered the programmer asleep at his terminal. He had been programming all night. (James, 1986).

1. Introduction

According to some researchers, the average work time over the last 20 + years has risen significantly for everyone in the US (Darrah, Freeman, & English-Lueck, 2007; Schor, 1991) and has quite likely risen in other parts of the world as well. Arlie Hochschild even makes a point that, as a result of cultural values shift, for many employees work has become home while home has taken on the role formerly reserved for work (Hochschild, 1997). Between the polarities of the rise of work for pleasure and liberation from corporate boundaries (Negroponte, 1996) and yet another step in capitalism taking over the employee's life (Agger, 2004), clearly significant changes are on the way in balancing work and private time.

Although other studies argue that some groups have experienced an increase in their leisure time (Robinson & Godbey, 1999), it is clear that—especially in case of knowledge-intensive jobs—work-related time demands have escalated (Barley, 1997; Fligstein & Sharone, 2002; Hassan & Purser, 2007). The bifurcation of work time is advancing (Jacobs & Gerson, 2004); more and more employees have to make the choice of either working very long hours or not working at all.

As a result of the industrial revolution, blue-collar workers and lower level white-collar workers have been able to secure the rights to limit the time they spend at work. Although initially introducing schedules served to discipline workers and increase their commitment, in the end it "liberated" them by defining their remaining time as free (Zerubavel, 1981; Zuboff, 1988). However, for managers and professionals, this process never took place. In fact, as Robert Reich points out, "The richer you are, the more likely it is that you are putting in long and harried hours at work" (2000, p. 5).

Still, contemporary consulting and management theory abounds in books, strongly emphasizing the importance of intellectual capital as well as the primary role it plays in gaining a competitive advantage and the crucial function of knowledge (and knowledge workers) for the company's well-being (Bell, 1973, 1989; Drucker, 1993; Hippel, 1988; Mallet, 1975; Senge, 1990; Stewart, 1997; Styhre & Sundgren, 2005). However, it does not go in hand with lowering the demands and stress put on the worker. Actually, in many companies this approach to intellectual work is merely propagandist and leads to an ironic paradox: On one hand knowledge-workers are depicted as the most valuable members of the company, while on the other they are under the highest pressure of impossible deadlines. Such a split between the official managerial discourse and the everyday practice is certainly not novel (Grant, Keenoy, & Oswick, 1998; Höpfl, 1995; Knights & Willmott, 1999). Still, in high-tech corporations it is particularly conspicuous.

Software and hardware engineers work in a practically permanent time shortage (Cooper, 2000; Jemielniak, 2007; Kunda, 1992). Not surprisingly, schedule slips in software engineering are extremely common (Connel, 2001; Humphrey, 1997; Kesteloot, 2003). Only 26% of software projects are completed on schedule, within budget, and with the required functions (Smith & Keil, 2003), while 63% suffer time overruns (Standish Group International "Extreme Chaos" report, 2001). Moreover, software development is a field of high distrust among the clients, programmers, and project leaders (Latusek & Jemielniak, 2007). In addition, high-tech employees often form a very specific occupational culture of their own (Bucciarelli, 1988; Garsten, 1994; Hertzum, 2002; Kraft, 1977; Kunda & Van Maanen, 1999; Trice & Beyer, 1993; Vallas, 2003), which highlights the contrast between management and the workers even more intensely.

Such issues makes the high-tech environment a unique and particularly interesting organizational setting (Kunda & Van Maanen, 1999), especially for the study of time constraint perceptions.

2. Research gap

Setting reasonable deadlines has been always considered crucial, and still extremely difficult part of software development (Brooks, 1975/1995). Unrealistic planning became a subject of expert's scrutiny nearly two decades ago (Boehm, 1991), and so did the models for estimating budgets and dead-

lines (Kemerer, 1987). Also, the signs of IS projects failure, including poor scheduling and timing, have been pinpointed and analyzed for at least over the last dozen years (Capers, 1995; Pollack, 2007; Reel, 1999), as well as good managerial practices for dealing with a troubled IT project (Keil & Robey, 1999). In fact, even now many authors agree that inevitable delays result from the very nature of programming (Connel, 2001; Kesteloot, 2003), as precise planning and estimation of tasks' difficulty level is not really possible in most cases. This is so, because the number of unexpected obstacles and slips is extremely high. Also, the complexity of tasks even for experienced programmers is hard to measure, as in each case coding and debugging take only a small part of the total time of the project. Joining modules written by different people, creating interfaces with other systems and programs, encountering undocumented behaviors of the environment, miscommunication with the customer, and many others, add to the picture of software projects being late by definition.

Apart from the technical setting, also social aspects of timing in programming have been investigated (Perry, Staudenmayer, & Votta, 1994). In fact, the organizational causes of delays from the schedule were sometimes found to be most typical in software development (Genuchten, 1991). Also, a number of authors studied the understanding of time in organizations, both from theoretical, and the research-based framework (Adam, 2004; Czarniawska, 2004; Hassan & Purser, 2007; Robinson & Godbey, 1999; Sabelis, 2001). There have also been many attempts to describe the social context of time pressure in high-tech (Kunda, 1992), as well as the studies of temporal shortages in software engineering (Perlow, 1997; Perlow, 1998), the differences in time management in software projects across cultures (Perlow, 2001), the work-family balance in high-tech environment analyzes (Hochschild, 1997), or programmers' temporal orientation on task-time rather than clock-time (Westenholz, 2006). Software development has often been presented as a type of knowledge work, difficult to control in a traditional sense, and managed by means of ideological and peer control (Czarniawska-Joerges, 1988; Kunda, 1992).

However, surprisingly, even though the use of time is clearly pivotal in controlling software development (as it affects both the length and the overall cost of projects), the connection between time and methods of normative or peer control have not been analyzed to its full extent. In particular, so far there have not been studies focusing especially on the software engineers' social construction of timing and delays, and especially on the perception of interruptions (from peers and from the management) and the symbolic use of time in the process of programming. Such a study, though, may be interesting both for organizational scholars, and for the specialists in software development.

Current literature shows that normative and peer control pressures employees to work long hours (Hochschild, 1997; Kunda, 1992; Perlow, 1997; Perlow, 1998). However, it is highly arguable if working long hours results in more projects being completed according to their schedule. This is a paradox involving time, which has not been studied so far in depth.

Thus, discovering the programmers' own (and non-determined by research design) understanding of timing and interruptions may shed new light on IT projects' social logic. Even if it does not reveal new facts about them, such research can allow a better insight into organizational dynamics and culture, which in the end affect also the very real project implementation. After all, apart from some objective constraints and circumstances in software, its also very important to realize what a crucial group of people engaged in a venture thinks about the recurring issue of exceeding the schedule, how they enact time, and how they construct what is disturbing to their work, and what is not.

Researching the symbolic meanings and values assigned to time by key actors of software development (software engineers and managers), especially in context of currently dominant normative control theory, may help in resolving the described paradox. Thus, this paper addresses the call to use more integrative and interpretive approach into time analysis of IT environment (Sahay, 1997). The objective of this article is to present the social construction of timing and schedules from the point of view of software engineers. The problems, discussed subsequently, are questions of delays and interruptions in IT projects. As described above, slips in schedules frequently occur in software development, and it may be interesting to learn the organizational logic behind them, according to the key actors involved. The field material serves as grounds for potentially more comprehensive future analysis of changes in social construction and enactment of time in modern workplaces.

3. Methods

This paper examines the results of a longitudinal ethnographical study conducted in a Polish IT company (codenamed "MicroMachine"). The study took place from September 2006 to February 2007, and included 4 months of non-participant observation onsite, 4 days a week in office hours (with occasional days missed, as well as 2 weeks off during the winter break). It was later on followed by further studies of similar businesses in the US, not covered by the presented article.

The researcher chose qualitative methods as particularly useful for the analysis of cultural perceptions among organizational actors (Burawoy, 1979; Czarniawska-Joerges, 1992; Denzin & Lincoln, 2003; Golden-Biddle & Locke, 1997; Schwartzman, 1993). The methods included non-participant direct observations, shadowing, and open unstructured interviews—each lasting approximately 60 min; the sample included 21 software engineers and 4 managers. All interviews were recorded and transcribed, with interviewees' consent. The anonymity of informants has been secured both by changing the name of the company and by codenaming the studied people. For obvious reasons, the presented few excerpts are but a small fraction of the collected material and were chosen as particularly exemplary to the ones omitted because of length limitations. Half of the interviewees were part of the team the researcher was spending most time with during the observations onsite, and as a result they were very familiar with the interviewer. Apart from the recorded interviews, many informal conversations with the members of this team took place (and were noted down from memory in the fieldnotes), helping to understand the organizational logic and determining, to wide extend, the reading of the interviews themselves.

The study was a non-directed research, conducted inductively. As a result, it began without a set of initial hypotheses and was aimed at defining the areas most important for social actors in the very process of study; it subsequently narrowed the findings in the following iterations. Four rounds of categories' coding took place before the field material began to stabilize. The study was an introductory research round to organizational ethnography (Atkinson, 2001; Denzin & Lincoln, 2003; Hammersley & Atkinson, 1995; Kostera, 2007; Schwartzman, 1993; Van Maanen, 1988). Coding was used for the convenience of the ethnographer, and not as a tool for generating a theory (Glaser & Strauss, 1967). The interviews were loosely directed (Whyte & Whyte, 1984) and were evolving around the subject of workplace, opinions on projects, managers, career, and timing. Thus, the presented study should be treated as an interpretive case study (Klein & Myers, 1999; Walsham, 1993) – a method particularly useful for the analysis of new concepts and phenomena, allowing a "sensitization" of fieldwork within the framework of existing theories.

Klein & Myers (1999) piece on interpretive research in IT field emphasizes the need to be sensitive to multiple interpretations and to include views from different work groups. The presented study takes this into account, although it relies more on the point of view of programmers. This is so because the purpose of the study is also to understand the point of view of a particular occupational group. Thus, the perspective of managers is acknowledged, but it only seconds and mirrors software engineers' concerns and observations.

The study is performative, not ostensive (Latour, 1986), as it aims to understand and explain the point of view of the interviewees (Greenwood & Levin, 1998), rather than to offer an ultimate interpretation of the analyzed problem "in reality." Considering the level of misunderstanding between the software engineers and managers, looking at the perception of workers, more rarely discussed in management and organization theory literature (being quite often biased in favor of managerial approach) may be worthwhile. This is particularly important, when considered that the group-shared stereotypes form reenactments of actors' organizational roles; they are the stories fundamental for workplace understanding (Berger & Luckman, 1967; Boje, 1991; Feldman & Skölberg, 2004; Weick, 1969). As such, despite being worth scholarly attention, they rarely are analyzed with proper respect.

Although the presented views from one profession may certainly be biased to some extent, even these workplace biases, are an important part of organizational reality and have huge influence on the process of work (Gill, 2003; Macrae, Stangor, & Hewstones, 1996). The alternative views and assumptions of the managers are used mainly to triangulate the outcome and widen the data interpretation (Blaikie, 1991; Kleine, 1990).

Understanding the ways software engineers sense-make schedules and construct delays, even if they are biased, is essential for widening our understanding of timing in IT. In this sense, the objective of this paper is not to discover "how it actually is" in terms of timing and schedules in software development, but rather to describe the perception of the key organizational actors. Thus, this paper concentrates on this occupation's perception, although it does include the managerial view to some extent.

MicroMachine is a medium sized company (ca. 200 employees), owning an office building near Warsaw. It is a limited liability company, with one major shareholder (with background in engineering, but not in computer science). It is providing B2B solutions mainly for the banking sector. It develops customized software systems, some of their own design, and some licensed from global players. The nature of software required by banks makes MicroMachine partially immune to competition from abroad: they offer local implementations and support, and also understand the Polish market demands really well. However, this makes them also practically non-competitive on the international scale, at least with the current product portfolio. Although MicroMachine occasionally gets contracts from completely new clients, most of their business relies on carefully cultivated contacts with banking and financial institutions. Even though MicroMachine is a very local player, it benefits from its location: Warsaw University has one of the best computer science departments in the world[1] and many of MicroMachine's programmers have graduated from there.

4. Findings

MicroMachine is located in the suburbs. In many respects it presents itself as a "fun" place: there are foosball tables in the hallway, free snacks, coffee and sodas are offered on each of the three floors, many employees sport iPods (using them also during work), the cubicles are decorated with posters, Dilbert comic strips, and almost all programmers wear jeans and T-shirts. And yet, MicroMachine's workplace is everything but peaceful and quiet.

My general impression upon arrival was that I had apparently landed in the center of a battlefield. People tapping on computers, discussing lively, frantically moving from one station to another, checking something. After waiting a couple of minutes, I finally approached the discussion group, introduced myself, and asked if they were busy. My suspicions were promptly confirmed: The whole team was working on a project for some important client, and they were indeed exceptionally occupied. It seemed only natural to ask when they thought it would be more convenient to come and talk. They unanimously suggested coming back in two weeks, to which I eagerly consented. Two weeks passed, and I returned to the same room. To my surprise, the teamwork seemed to me even more hectic. As somebody told me, a serious crash had occurred in a banking system they supported.

I dared not interrupt such an intense work session. Instead I decided I might as well hang around at the company, make observations, and speak to people from other departments. I also made sure to ask when a lighter period was expected and was informed that, if I showed up after another two weeks, I would easily find the situation back to normal. When I returned after two weeks, I finally learned an important lesson, which was subsequently verified in other companies, too. The slack period was always to come—after Christmas, after the current project, or sometimes just soon. Quite unsurprisingly, though, it never did. Instead of asking when there will be more time, I started making appointments with individuals and spent the remaining time conducting observations.

Still, in the case of the interviews that were scheduled ahead, many software engineers requested a postponement at least once. Eventually it turned out to be easier to approach a given programmer in the morning and ask for the interview to take place during his or her lunch break.

Most software engineers worked at least until 6:00 or 7:00 pm. Some, as observed, were staying longer into the night, and a few regularly appeared at work at 7:00 am to leave as early as around 5:00 pm. A 63-h workweek was the average norm. In times of crises or just before releasing a new product, one of which appeared every couple of weeks, the team members worked even more, up

[1] E.g. for several years it has been No. 1 computer science school in the TopCoder ranking (http://www.topcoder.com/stat?c=school_avg_rating).

to 80 h a week in a couple of extreme cases observed. As several of the programmers succinctly and identically put it "there is never enough time." One software engineer states rather philosophically:

> [MiniC7:] It is a result of the very essence of reality that software projects always take longer and cost more than it was estimated. In the beginning, there is a very general idea. Then some changes are agreed upon, but nobody knows what the system should do. The user has some illusions that the firm. . .I mean the user has some general knowledge how computers work and that many nice things could be done, but he doesn't realize that he has to precisely articulate, really precisely tell what he wants.. . .some problems will arise, which he couldn't even imagine, and that actually he is the one who should know how to deal with them.

Most interviewees shared the belief that exceeding the schedule is only natural—a norm rather than an aberration:

> [Sunny5:] Barely any deadline is respected. The task is never as easy as it seems in the beginning. The general problem is usually trivial, then the time is set, but when it comes to details, you have to take all the possibilities into account and you have to do something about it. Sometimes the language also hampers the work, some things simply cannot be done. Sometimes, gosh, may it be rare, the user recalls something and changes the entire concept or makes a condition that spoils all the work.

Although MicroMachine was proud of its flexible schedules, which allowed e.g. two of the programmers to pursue their graduate degrees part-time, in practice all software engineers worked for at least 45–50 h per week even in the most slack periods.

4.1. Time out

The perception of workloads, dedication, etc., was diametrically different in the eyes of the managers. As one of them observed (quite representative of the rest of the interviewees from this group):

> [Comp11:] Here in the company sometimes I see [programmers working], especially when I return home in the evening, and some sit and are somehow closed and would prefer not to leave at all. There was a programmer I knew. . .Once I was going home from work, then the next day I returned to work, and he just said, "Jesus Christ, I forgot to leave." Well, but he was seeking solutions. Anyway, he was a really good man; he actually is making a career here, in Warsaw.

The manager told about a case in which the described software engineer was so engrossed by programming that he forgot to leave work. Although the manager was not particularly praising such attitude (which happens in similar businesses—compare Perlow, 1998), he perceived the extreme workaholism as a matter of personal preference of the programmers, rather than of the company's explicit demands. Indeed, many of the managers referred to programmers as people who love to work, even competing as to who puts in the most effort. The managers seemed to believe that software engineers may also be showing off by spending more and more time at work, which was in clear opposition to what the programmers said themselves.

In fact, most of the interviewees expressed sheer repulsion toward working long hours and "tormenting people" with temporal demands, which is in dire opposition to what the interviewed managers believed. Time management was also one of the most renounced managerial roles. Hurrying and asking for faster delivery was pointed out as the single most despised treats of a manager.

According to the company's policies at MicroMachine, employees enjoyed a lot of freedom in defining their own schedules, as typically it was the programmer who was to suggest on how time-consuming a given task would be. In practice, however, it did not change much. All programming tasks were, according to the interviewees, highly unpredictable anyway. Therefore their assumptions and estimations relied heavily on the management and the customers' expectations. A couple of software engineers confessed that they are trying to guess what timeline their boss thinks of, rather than really calculate the amount of work needed for the assignment, simply because they found any estimates

too vague and contingent. Thus, they were even more detesting these managers, who tried to "motivate" them by emphasizing the need to "do things quicker".

The programmers—even those who were happy with their workload and family-work balance—shared two things in common: they all worked quite a lot and none liked temporal boundaries imposed by management. In fact, many actually shared heroic stories of programmers who outfoxed managers in avoiding pressure. The following story is particularly exemplary to the attitude of many programmers:

> I heard about this guy who was the software analyst here a couple of years ago, and he had a pretty independent position. He was pretty smart. He always showed this "job goes first" approach, you know. But in fact he was beyond that, he laughed about it. And he had real guts! Once he decided he needed a bit of vacation. And it was straight after his boss changed, it was unlikely the new one would ok his absence, and so on, you know how it is. So this guy came to work on Wednesday, logged into the network in his cubicle, opened a couple of applications, left his jacket on the chair, put a cup of water on the table, and just drove with his wife to the mountains till Monday. The funniest thing is that nobody really noticed he was not there. I mean, probably some people knew he was not around, but nobody said anything and the manager thought he's just hard working somewhere else.

Although this story is hard to believe and quite likely is a local legend, such myths define how the organizational workplace is being constructed. In the story presented above, the brave engineer was able to get a secret vacation while still managing to look as if he worked hard at the company. Several other programmers shared accounts similar in implication. Other reported strategies included e.g. making an impression of being on the company's premises by leaving a car in the parking lot, or forwarding desk phone to a mobile.

Being available to organizational demands is valued: many of the interviewed were expected to be available by phone just in case, even when on vacation. The following story of a project crisis is quite exemplary:

> [Peter]: It was a major crash, not that the bank stopped, but their daily backup system repetitively hung up during backups, which is something not really desirable in a bank branch, if you get my point. It was something nobody can predict. [Such crashes] are very rare. I mean maybe they're not very rare, but this time nobody knew what happened, why, and so we didn't even have any ideas what to amend. It is one of our major clients, so we really had to do something even though it was not really certain if it's actually the program that's at fault. We still don't really know if it was, we just managed to bring it back to normality, that's what matters. I was actually not at work then, I had a couple of days off, sailing. I had a cell phone, we usually keep them on just in case, even though we normally don't have to use them; it is exactly for situations like this, you know. So I spent a couple of hours discussing [it] with guys from the office and they managed to find a workaround, so there was no need for me to come back.

Interestingly enough, Peter himself did not really believe his participation was really necessary. He was happy, though, that he was not expected to come back from his vacation, and a phone consult was just enough.

Many others made a point that the situation was worse "elsewhere" (in other software companies they heard about or just in a stereotypical IT business): People are tormented by constant pressure to work all the time. Although the temporal burden was quite heavy in the companies for which they worked, apparently these interviewees considered it to be comparatively lighter. They may have had good reasons for such perceptions; stories from other IT companies (Perlow, 2003) indicate that some software corporations manage to impose even heavier demands on their employees.

Most of the interviewees had plenty of stories to share about how they had to work extremely long hours on a project; most of them were unable to recall a situation in which they unexpectedly finished their job early and had nothing else to do. As one programmer stated:

> [Radius4:] Oh, last year I was able to write a program in just three days, but I slept only for four hours, on the first day. I sent the program, went home, and was only hoping not to fall asleep on

> the way back. It is difficult to say if I wanted to work this way or if there was such a need or [something] else.

Such style of working—although surely allowing strong focus and concentration on one project—was also quite likely ineffective to some extent. We can safely assume that the intellectual abilities lower significantly when one gets tired.

Interestingly, even the managers with prior extensive programming experience, also seemed to have very different perceptions toward time pressures and unrealistic deadlines in comparison to their former peers. It would appear that the role or task performed, rather than knowledge or experience were determining this difference, which was particularly visible when observing the reception of interruptions in programming work.

4.2. Interruptions

In fact, focus—especially in coding—is very difficult to maintain. Along with the extreme pressure related to time described above, it is very surprising that the shadowed software engineers in the study spent as much as half of their work time discussing issues with their peers. Most often software engineers asked their colleagues about some part of the code on which they were working, problems they encountered, or general programming issues.

The level of commitment to helping others varied significantly among the observed. Most helpful programmers spent a lot of their social time on conversations that were not initiated by them, while several programmers "leeched" the help of others, spending most of their time with others in communication started solely by them.

Still, none of them expressed the belief that inquiries from their colleagues are disruptive. In fact, most software engineers said that cooperation with others is essential in their work. Those who spent less time helping others than getting help were more eager to say that other programmers are distracting them. Nevertheless, they valued peer support and insisted on its necessity, believing that information from colleagues is more reliable than from other sources. In programming, as it turns out, the easiest way to approach an unresolved problem is to look for its solution on specialist forums and, if it does not help, ask colleagues if they have heard about the issue before. Occasionally, this means asking friends who work for the competition; in three such cases observed during the research, programmers explained that they do not perceive asking friends who work for the competition as problematic:

> You know how it is: maybe I reveal a little bit of what we're doing here by even just asking the question, but I try to be rather brief in that. But the point is that it is for the company's own good that I am able to find the solution quicker. Also, it works like that—he scratches my back, I scratch his, so there is sort of an exchange from time to time. Everybody wins.

A vast majority of all interviewed programmers pointed to their peers as the best source of information in case of trouble. The requests for help were regular; occasionally they were, in fact, just for socializing (as noted down in the field notes):

> Paul leans back in his chair, still looking into the screen. "Hey, I wonder if I should check for this php backdoor hole again," he says. "Sure, just wait until I enable shell commands through URL," merrily throws off Stan, looking into his monitor and typing something simultaneously. They both laugh.

Both programmers later confessed that they knew that no need whatsoever existed to look for any backdoor holes. A manager had come to them during lunch and apparently wanted to show he also had some technical knowledge; he requested a security checkup on "php backdoor holes," trying to sound authoritative and knowledgeable. This became a topic of jokes for a little while. Many other conversations seemed to serve reassuring or conventional purposes. Still, the support provided for one another by programmers was enormous (both in their own description and as noted during observations). However, not a single interviewee mentioned that he minded such interruptions.

The situation differed dramatically when the interviewees talked about being approached by other people—namely, managers and clients asking for support. One of the interviewees used a dramatic comparison:

> [Q:] What are the situations you dislike about your work most?
> [Comp10:] You want the whole list? [snigger]. Well, but seriously...I think I hate nagging by the managers most. I mean, for God's sake, if I am to report on how I am proceeding with the project all the time, when exactly am I supposed to do the real work? It is as if a surgeon was supposed to conduct an open heart operation, but at the same time fill in the paperwork describing what he is doing, and also report every 10 min about his status to the ward.

In fact, apart from hurrying, the most disliked managerial behavior among the interviewees were status requests:

> [Q:] What kinds of things you dislike most in your manager?
> [A:] Aaah... Probably, asking how is it going all the time. You see, when I work, I want to work. If every 15 min somebody comes and asks how am I doing, and when the project will be ready, and if everything is fine, and if I could come to a "very short meeting" [interviewee gestures quotation marks], you know, then of course that nothing will be fine, because I won't be able to concentrate on the job itself.

Most of the interviewees complained that seemingly minor requests for status reports were exceedingly disruptive; many of the software engineers felt that, after being approached, especially by a manager, they found it very difficult to go back to their work. Without any doubt, among the many causes of programming efficiency decline, spontaneous interruptions play a major role.

Consequently, many of the interviewed programmers resorted to shunning strategies. Some shifted working hours, either by coming to work early in the morning or by coming and staying significantly later[2]. Some purposefully unplugged their phones or even blatantly lied about their commitments elsewhere to avoid attending meetings (as one of them told me, "If you attended all the meetings they call you in, you probably could become a very respected manager pretty easily, the only problem being you wouldn't do any work whatsoever").

Among the things the interviewees mentioned as most frustrating and bothersome at work were invitations to come to meetings, phone calls, and status checks; many programmers tried to make their schedule at least partially incompatible with the regular business hours for exactly such a reason.

4.3. Discussion: normative control

Clearly, the studied processes can be partly explained by theories of normative control, which is much discussed in the literature. Under its rules "discipline is not based on explicit supervision and reward, but rather on peer pressure and more crucially, internalized standards of performance" (Kunda, 1992, p. 90). Thus, the mainstream explanation of the described phenomena is that IT companies try to monopolize their employees' minds while the managerial discourse reinforces the role of the programmer as that of a passionate geek. This stems from the fact that organizations relying on work that is not reducible to describable routines and behavioral patterns increasingly try to at least secure their workers' loyalty (Hochschild, 1983; Kidder, 1981). Lewis Coser labels such firms as *greedy* for the workers' dedication and time (Coser, 1974). Such organizations are by no means rare in high-tech environments. It is assumed that when the workers reach their limits with every effort, top efficiency is also achieved. The capitalist seeks to create the most surplus value he can by extending the working day to absorb the greatest possible amount of surplus labor.

In this respect, it is pretty understandable why those interviewees who theoretically worked part time in fact were also inundated with tasks. Rosabeth M. Kanter wrote that "the organization's demands for a diffuse commitment from managers is another way to find concrete measures of trust,

[2] Zerubavel (1981:65) writes that temporal exclusion from society may be a severe punishment, commonly used among monks in the Middle Ages. The "graveyard shift" was not perceived as uncomfortable to the interviewees, though, perhaps because it occurred only by their clear choice and usually meant a couple of hours rather than the whole night.

loyalty, and performance in the face of uncertainties" (Kanter, 1977, p. 65). The same thing may be said of programmers. As a result, "managers and professionals (particularly engineers) are those who most closely identify with the companies for whom they work" (Kunda & Van Maanen, 1999, p. 64).

Managers discovered that increased dedication and imposition of a value system that appraises work highly makes workers more productive. People who love their jobs are more effective; therefore, in many organizations the role of management is (even if implicitly) understood as making them really affectionate about work or "allowing the software engineers to follow their bliss," as one of the managers put it. In this light, the theoretical incongruence of normative control with knowledge work literature disappears. Knowledge workers are supposed to be excited about what they do, as this is what defines their occupation as well. Therefore, it is not the conscious or planned pressure from the organization that forces commitment, but rather the strong underlying expectation toward the organizational role. This is why, even in software companies that do not push it to extremes, time of work is practically never limited to the time-clock and is organized cyclically around projects (Shih, 2004), tending to occupy most of the employees' time.

Paradoxically, just like with liberation from schedules, the strict control over the physical aspects of blue-collar work present in many neotaylorist and mcdonaldized workplaces to some extent emancipates workers from emotional control. Reducing humans to the machine level actually allows them to keep their humanity intact and mind unaltered. However, this cannot be so in the case of knowledge-workers, in which the individual norms—the heart and identity—have to bend to the will of the organization. It is assumed that "workers owe not only a hard day's work to the corporation but also their demeanor and affections" (Edwards, 1979, p. 148). Thus, software corporations need to become what Goffman (1961) calls "total institutions", limiting the time spent outside them and making the behaviors and perceptions of inmates uniform. Whether feelings and roles ordained on someone are "real" or "just perfunctory" is another question, but emphasizing internal motivation is believed in and commonly used. Most interviewed managers used euphemisms such as "fitting to the culture," "being a part of the team," and "sparking with the others" to describe the desirable features of a software engineer. These further reflect the extent to which a worker is eager to accept the workaholic environment.

Such instant pressure indeed forces the idea of doing one's best; however, by definition it cannot leave slack the reserve of time. As software projects are highly changeable and unpredictable (according to interviewees, this is mostly due to, among others, technological difficulty, changing requirements, misunderstandings with the clients, as well as unrealistic promises made to win contracts), such an approach leaves no margin for the periods when things go wrong. As a result, being even slightly unrealistic in management may lead to huge setbacks (Brooks, 1995; Kesteloot, 2003). Although individuals can work 60- to 70-h weeks, it is difficult to occasionally intensify efforts; it is barely possible to add an additional 20 to 30 h when needed. Moreover, in the IT field, adding new people to the project rarely helps. As Brooks (1995) observes, software projects are "late by design", which explains not only why so many projects are behind schedule, but also why organizations—and even customers—readily accept the situation. In their roles, software engineers accept delays much more easily than disloyalty. They dislike time constraints, as they are correct to perceive them as manipulative. No wonder that time constraints are considered to be the most de-motivating factors for software practitioners (Baddoo & Hall, 2003).

4.4. Discussion: ritual time sacrifices

However, within the above theoretical framework, some of the observed phenomena are difficult to explain. For example, if all employees are working with full commitment, it is difficult to understand why are the delays and project failures so common in software industry. More interestingly, if the interruptions from managers are manifestations of power and attempts to exert more pressure, just as interruptions from peers are tools of concertive (Barker, 1993) supervision – they both serve the same purpose of reinforcing normative control. Their function is to set stricter norms of performance and to increase organizational/occupational loyalty and dedication. If it is so, then especially under extreme time pressure, it is totally unclear why is the reception of interruption from managers and from peers so diametrically different. Therefore I would like to propose understanding time as a form of

symbolic organizational currency, used differently in relations with the managers, and differently with peers.

Surprisingly, in many new technology projects problems are recognized pretty early, but are not solved until they become most urgent (Van de Ven & Polley, 1992); the study suggests that this results from software development encouraging the culture of "individual heroics" (Perlow, 1997). This is quite understandable given that, in knowledge-intensive jobs, direct supervision does not help at all. To the bystander, an effective, successful programmer (or, for that matter, a lawyer) looks exactly like the one who is a failure and sluggard. Only the final results of work can be to some extent measured, while the very process of labor is a black box for anyone not directly involved. Besides Microsoft Office Project/Trac, meetings with white board pages, and oral (as well as written) status reports, the managers do not have other tools to push and control their subordinates. Thus overt leadership must be substituted with a covert approach, management with counsel, supervision with inspiration (Leidner, 1993).

As a result of the lack of other criteria, employees are praised for making visible sacrifices—e.g., by ostentatiously giving their work the highest priority. Companies have trouble evaluating their software engineers' performance. Controlling time at work, as well as praising individual heroism, is much easier. From the corporation's point of view, time turns out to be the most sensible value denominator for appraising efficiency and individual progress.

Coming back from vacations, canceling important family events, or staying at work overnight are clear, tangible signs of effort and loyalty; they have an invaluable (and irreplaceable) symbolic value. Taking a cell phone on vacations is a "natural" and unquestioned symbol of prioritizing work highly. The company may show its humane face by not calling too often, but the boundary between private and professional life is certainly not cultivated. Leaving one's car in the company parking lot and returning home by cab or leaving the computer logged into the company's network for a night are gestures that suddenly make more sense in such a "time famine" environment (Perlow, 1999). They are clear signs of organizations' recognition of time (rather than work result) as the common value denominator. As a consequence of placing so much emphasis on time and its importance, higher than of any other inputs, its ceremonial role becomes pivotal. Although time rituals have always played an important role in contemporary societies (Gingrich, 1994; Rappaport, 1992), now we are observing an interesting social change in which time, and not the effect, becomes the key symbolic contribution one can give (Agger, 2004).

In fact, in the observed crisis, Peter who was responsible for the software that crashed in a bank, was not only not punished or rebuked for the program malfunction, but was later praised for the fact that he dealt with the problem remotely—even though he confessed that his help for the team was minimal. What was appreciated was the fact that he demonstrated his dedication by staying on the phone and sparing some of his vacation time for work. For this, slips in schedules and crashes are good tests. Loyalty can be proven only when one has to choose between work and other aspects of life. This is why part-time work in knowledge-intensive companies is barely possible: just like under the rules of normative control, but offerings of time become ceremonial.

Also, the symbolic role of time widens the explanation of reasons why so many IT professionals prefer to become contractors. In spite of other determinants, many software engineers simply desire to gain sovereignty over their time so strongly that they are willing to give up job security. They chose artistic autonomy over dependence of a corporate engineer (Jemielniak, 2008). As Barley and Kunda (2004) convincingly show, once they go freelance, they quite often find that they lose even more control. Nevertheless, the freedom from schedules and (even if illusionary) reign over time are important reasons driving software engineers to go independent.

In this sense such ritualistic sacrifices may be more appreciated than finishing work on schedule. A software engineer who sticks to the deadline might have been assigned too little work, might have not done his/her work thoroughly, and so on. Only working all the time is an automatic proof of maximum effort; thus, going beyond the schedule is also a sign of hard work. In this light, even without the need for normative control, the methods of evaluation of work in software engineering naturally favor those who make visible commitments. "Impression management" becomes of key importance (Goffman, 1959/2000). The legend of a software analyst who took vacation without the manager even noticing is symptomatic. Such organizational myths have major significance: they help to decide what is

acceptable and what is not; they describe which behaviors are smart and which are stupid (Bowles, 1989; Gabriel, 2004; Hatch, Kostera, & Kozminski, 2005). In this story the hero was able to get two valuable prices simultaneously, as he spent a couple of days just having fun skiing and still managed to instill the image of a hard-working engineer.

He doubled his paycheck in symbolic time currency, as he effectively achieved the contemporary ideal of time compression by "being" in two places at once and performing tasks in zero duration (Adam, 2006). In this sense, the interruptions from managers are detested because they limit the scope for heroism. After all, the more software development is under control, the more it is difficult to save the day. As organizations tend to appreciate software engineers most in the times of crises, it should not be surprising that the social setting actually encourages those.

One of the paradoxes of communication technology advances is that eliminating communication delays, while in theory good, inevitably leads also to an increase in work interruptions (Rennecker & Godwin, 2005). This is particularly acute in situation of software developers, who have to translate between the needs of end-users and the expectations from the management, which quite often are contradictory (Howcroft & Wilson, 2003). The software engineers' preference for odd hours stemmed from the fact they could avoid being pestered by managers and clients. Still, what was most unusual was the fact that they did not perceive interruptions from their colleagues as disruptive. They were also quite eager to engage in small talk and socialize with peers either in person or through IMs. In a different study, about 40% of the employees did not resume the work they were at before the interruption (O'Conaill and Frohlich, 1995). What is striking, though, is the fact that here only the conversations initiated by the management were clearly perceived as distracting (for different results see e.g.: Perlow, 1997). This was probably to a significant extent due to the fact that they had to rely on each other; in many cases, they encountered problems that were not described in official manuals or instructions, similar to the situation facing technicians from Julian Orr's study (Orr, 1996). As a result, peers were perceived as the most valuable source of information (compare Hertzum, 2002; Hippel, 1988; Kraut & Streeter, 1995).

Naturally, interruptions from peers base much more on a symmetrical exchange of support (as well as of exchange of friendly gestures and tokens of solidarity, in opposition to the management), and are no threat to the heroic role. Even more importantly, peer interruptions are clearly not tools of concertive control. Just the contrary: they instill the system of professional trust and mutual dependence. Software engineers ask colleagues for help not to subdue them, but to recognize the value of their knowledge. The approached supporters are then able not only to give the desired information itself, but also to repay their peers with the symbolic currency most valued by their companies: time. Thus, peer interruptions are not unwelcome also because they are in diametrical opposition to the rule of normative control.

5. Summary and study limitations

Employees who are pushed to full dedication and are instantly observed in a Panopticon-like (Foucault, 1975/1993) labyrinth of cubicles tend to develop a counter-culture of their own, resisting pressure—although not openly. As described, the legends of heroes who outsmarted the evil managers are widely distributed, and the management, in general, is a common subject of jeers and occupational satire. In a climate of distrust and animosity, all estimations are doubly erroneous, as the employees do realize that once they finish some task, they will be assigned another one and thus take additional precautions not to be overly optimistic in planning; on the other hand, some programmers instead of trying to actually estimate the time needed for work, prefer to guess the superiors' expectations. Additionally, the managers tend to be suspicious about the professional estimates given by their subordinates and press for quicker outcomes, in spite of their own ignorance of the subject.

The presented study shows that in such an environment in which time is a very scarce resource as well as a commodity used widely for symbolic contributions to the employing organization, interruptions from the side of management are perceived as exerting additional pressure and demanding even more sacrifices, and as such are perceived as extremely disturbing and inappropriate. However, this of course is not the case with interruptions from the side of the colleagues. Peer interruptions—whether

substantially technical or mainly for fun—are construed as community building and friendly (in opposition to managerial interventions, which are perceived as disruptive and unnecessary, if not plainly hostile).

As a result of such timing perception, programmers share knowledge and support each other much more gladly than the mainstream knowledge work theories would suggest (according to theory they could be reluctant to do so, to protect their own competitive edge and not to stretch the already strained schedule). Apparently, programming is an activity equally involving solitary and social activities (Kociatkiewicz & Kostera, 2003); for software engineers, the gift of time—so precious in the organizational setting—is a form of confirming the group's integrity and defining "the other" (the management). Regretful reactions towards superiors' intrusions contrast with welcoming of the breaks from the side of the peers, and all reinforce the occupational divisions and serve as an reenactment of the fundamental conflict of interests.

These processes of knowledge development and spontaneous creativity are both hampered by the assumptions about time imposed by contemporary economy and society (Hassan, 2003). The "time is money" paradigm is in clear opposition to the creative, nonlinear temporal chaos that programmers work in. The measurable clock time, traditionally perceived as an essential part of capitalistic control over the labor process (Thompson, 1967), reflected e.g. in PERT, Gantt chart, or different critical path methods (practically used also in the studied company, also through project management software), are not really applicable and compatible with software time, experienced by the programmers (Nandhakumar, 2002). One of Bolter's (1984) conclusions was that intense, extended contact with computer leads to breaking the temporal unilinearity. This can be extended to many other creative industries, but in software development is particularly visible: programmers often exist, as Ellen Ullman (1995) puts it, "out of time", totally captivated by their brainchild, barely aware of the outer world and time-flow. Thus, the subjective perception of time flow, commonly very much influenced by tasks and a job performed by an individual (Das, 1993), in case of software engineers is additionally under the strong influence of interactions through and with computers, as well as of the very nature of teamwork in programming. This is so, because IT organizations are dominated by project organization of work, and especially in software development they are often conditioned by the tempo and project's stage.

The described system of symbolic time contributions and demands present in software development clearly shows that in software development strict control by clock time is barely possible: in extreme it leads to pathologies such as hiding one's non-presence from the office, or evaluating programmers by the sheer amount of time spent at work, and not the value of its outcome. Peer interdependence is strictly connected with this process: apparently programmers prefer to self-organize their workplace, and (while satisfying their managers by playing the symbolic game) do their work in an alternative grid. Solidarity and professional reliance, instead of control, in case of IT projects allow better adjustment to software time, which is also why programmers do not perceive their colleagues as disruptive, whenever they interrupt their work.

In case of knowledge industry, the contradictions between capital and labor reflected in interactions between the managers and the software engineers are often less important (if even existing) than time enactment. It is the temporal perception, rather than class conflict, that sets the grounds for occupational tension. Time, used as a symbolic universal currency, becomes the key to understanding many of knowledge work phenomena.

The practical implications of this fact should not be underestimated. Clearly, the organizational culture and management practices promoting peer control and collaboration among programmers seem to be better adjusted to the split between clock time and software time. Also, organizations able to recognize symbolic character of some of the time-related behaviors of the actors should be more effective both in practical time management, and in eliminating delays resulting from "individual heroics" and related processes. Consequently, in organizations where performance is evaluated by the team, rather than by superiors, the risk of appraising just the amount of hours spent in the office, rather than usefulness of work, is definitely lower. Elimination of the managers from evaluation process could also improve the recognition of programmers most helpful to the others, but not necessarily most effective in their own share of work. Additionally, in peer-controlled workplace, the tasks perceived as peripheral and conducted in "intangible time" (as O'Carrol calls it: 2008), but in fact essential for program-

ming work, may be more appreciated and taken into account in planning. Finally, if temporal structures are social conventions, sometimes falsely identified with clock-time (Zerubavel, 1979), especially in project planning and monitoring it may be worthwhile to recognize their sense-making character (Orlikowski & Yates, 1999). Eliminating behaviors, such as interruptions at work, which are perceived as time-disruptive by software engineers, even though if they are not understood as such by their superiors, may make IT projects easier to conduct, although this issue clearly calls for further research.

Time slips belong, by definition, to managerial vocabulary, but they are important for the organization's interactions with its many stakeholders, and most importantly clients. Introduction of symmetrical peer control, instead of managerial supervision, may help in reducing the delays caused by symbolic time games played for the management. Similarly, although schedules are important and tracking deadlines is extremely useful (in Micromachines MS Project, as well as Trac wiki system was used for this purpose), project management software should be treated as a tool for communication between software engineers, manager and customers, rather than as medium of disciplinary control.

Additionally, managers responsible for IT projects need to recognize and acknowledge the different concepts, perceptions and uses of timing in software engineering. Understanding that time plays an important symbolic role in programming and high-tech workplace may in itself lead to improvements in communication. As a result, it may not only gradually better the image IT managers currently have in the eyes of software engineers, but also allow for smoother and more effective collaboration between them.

This paper aimed to show the symbolic dimension of schedules, delays and interruptions in software development. It has shown, on the example of a software company, how can time become a symbolic currency used by the key organizational actors. The study revealed how time can be valued more than plain results by the employing organizations and, how it can be a friendly contribution one can make to his/her peers (being also at least a formal admission of one's knowledge and usefulness, as at least theoretically only people who are considered able are asked for help).

The presented research was, by design, limited to the studied organization. The results, however, can be carefully generalized to other software companies and, more broadly, other knowledge workers. The paper shows that symbolic value of time, often overlooked in workplace analysis, plays crucial role in contemporary knowledge-work and constitutes one of its very distinctive features. Thus, the findings clearly call for further research, so as to determine the influence of symbolic time exchanges and contributions in other parts of knowledge-intensive industry.

Acknowledgements

I would like to thank Barbara Czarniawska, Davydd J. Greenwood, Robert Hassan, Monika Kostera, Gideon Kunda, Ida Sabelis, whose suggestions, directions and support were extremely useful at the early stages of paper writing. I am also indebted to Daniel Robey and the anonymous Reviewers, whose detailed and comprehensive reviews significantly helped in improving the article.

References

Adam, B. (2004). *Time*. Cambridge, UK – Malden, MA: Polity.
Adam, B. (2006). Time. *Theory, Culture and Society, 23*(2–3), 119–138.
Agger, B. (2004). *Speeding up fast capitalism: cultures, jobs, families, schools, bodies*. Boulder: Paradigm Pub.
Atkinson, P. (2001). *Handbook of ethnography*. London, Thousand Oaks, Calif: Sage.
Baddoo, N., & Hall, T. (2003). De-motivators for software process improvement: an analysis of practitioners' views. *Journal of Systems and Software, 66*(1), 23–33.
Barker, J. R. (1993). Tightening the Iron Cage: Concertive Control in Self-Managing Teams. *Administrative Science Quarterly, 38*(3), 408–437.
Barley, S. R. (1997). Foreword. In L. A. Perlow (Ed.), *Finding time. How corporations, individuals, and families can benefit from new work practices*. Ithaca-London: ILR Press.
Barley, S. R., & Kunda, G. (2004). *Gurus, hired guns, and warm bodies: itinerant experts in a knowledge economy*. Princeton, NJ: Princeton University Press.
Bell, D. (1973). *The coming of post-industrial society; a venture in social forecasting*. New York: Basic Books.
Bell, D. (1989). The third technological revolution and its possible socioeconomic consequences. *Dissent, 36*(2), 164–176.

Berger, P. L., & Luckman, T. (1967). *The social construction of reality; a treatise in the sociology of knowledge*. Garden City, NY: Doubleday.
Blaikie, N. W. H. (1991). A critique of the use of triangulation in social research. *Quality and Quantity, 25*, 115–136.
Boehm, B. W. (1991). Software risk management: Principles and practices. *IEEE Software*(January), 32–41.
Boje, D. M. (1991). The storytelling organization: A study of story performance in an office-supply firm. *Administrative Science Quarterly, 36*(1), 106–126.
Bolter, D. J. (1984). *Turing's man: Western culture in the computer age*. London: Duckworth.
Bowles, M. L. (1989). Myth, meaning and work organization. *Organization Studies, 10*(3), 405–421.
Brooks, F. P. (1975/1995). *The mythical man-month: Essays on software engineering* (Anniversary ed.). Reading, Mass: Addison-Wesley Pub. Co.
Bucciarelli, L. L. (1988). An ethnographic perspective on engineering design. *Design Studies, 9*(3), 159–178.
Burawoy, M. (1979). *Manufacturing consent: Changes in the labor process under monopoly capitalism*. Chicago: University of Chicago Press.
Capers, J. (1995). Patterns of Large Software Systems: Failure and Success. *Computer, 28*(3), 86–87.
Connel, C. (2001). Why software is (almost) always Late. (2007). <http://www.chc-3.com/talk/why_software_late.ppt> Retrieved 07.07.07.
Cooper, M. (2000). Being the "Go-To Guy": Fatherhood, masculinity, and the organization of work in silicon valley. *Qualitative Sociology, 23*(4), 379–408.
Coser, L. A. (1974). *Greedy institutions; patterns of undivided commitment*. New York: Free Press.
Czarniawska-Joerges, B. (1988). *Ideological control in nonideological organizations*. New York: Praeger.
Czarniawska-Joerges, B. (1992). *Exploring complex organizations: A cultural perspective*. Newbury Park, Calif: Sage Publications.
Czarniawska, B. (2004). On time, space, and action nets. *Organization, 11*(6), 773–791.
Darrah, C. N., Freeman, J. M., & English-Lueck, J. A. (2007). *Busier than ever! Why American families can't slow down*. Stanford, Calif: Stanford University Press.
Das, T. K. (1993). Time in management and organizational studies. *Time and Society, 2*, 267–274.
Denzin, N. K., & Lincoln, Y. S. (2003). *Strategies of qualitative inquiry* (2nd ed.). Thousand Oaks, Calif: Sage Publications.
Drucker, P. F. (1993). *The new society: the anatomy of industrial order*. New Brunswick, US: Transaction Publishers.
Edwards, R. (1979). *Contested terrain: the transformation of the workplace in the twentieth century*. New York: Basic Books.
Feldman, M. S., & Skölberg, K. (2004). Stories and the rhetoric of contrariety: Subtexts of organizing (change). *Culture and Organization, 8*(4), 275–292.
Fligstein, N., & Sharone, O. (2002). *Work in the Postindustrial Economy of California*. University of California Institute for Labor and Employment. <http://repositories.cdlib.org/ile/scl2002/FligsteinSharone>.
Foucault, M. (1975/1993). *Nadzorować i karać. Narodziny więzienia (Surveiller et punir. Naissance de la prison)*. Warszawa: Spacja.
Gabriel, Y. (2004). *Myths, stories, and organizations: Premodern narratives for our times*. Oxford – New York: Oxford University Press.
Garsten, C. (1994). *Apple world: Core and periphery in a transnational organizational culture*. Stockholm: Stockholm University Press.
Genuchten, M. v. (1991). Why is software late? An empirical study of reasons for delay in software development. *IEEE Transactions on Software Engineering, 17*, 582–590.
Gill, M. J. (2003). Biased against "them" more than "him": stereotype use in group-directed and individual-directed judgments. *Social Cognition, 21*(5), 321–348.
Gingrich, A. (1994). Time, ritual, and social experience. In K. Hastrup & P. Hervik (Eds.), *Social experience and anthropological knowledge* (pp. 166–179). London – New York: Routledge.
Glaser, B. G., & Strauss, A. L. (1967). *The discovery of grounded theory: Strategies for qualitative research*. Hawthorne, NY: Aldine de Gruyter.
Goffman, E. (1959/2000). *Człowiek w teatrze życia codziennego (The presentation of self in everyday life)*. Warszawa: KR.
Goffman, E. (1961). *Asylums; essays on the social situation of mental patients and other inmates*. Garden City, NY: Doubleday.
Golden-Biddle, K., & Locke, K. D. (1997). *Composing qualitative research*. Thousand Oaks, Calif: Sage Publications.
Grant, D., Keenoy, T., & Oswick, C. (1998). *Discourse and organization*. London, Thousand Oaks, Calif: Sage Publications.
Greenwood, D. J., & Levin, M. (1998). *Introduction to action research: social research for social change*. Thousand Oaks, Calif: Sage Publications.
Hammersley, M., & Atkinson, P. (1995). *Ethnography: Principles in practice* (2nd ed.). London – New York: Routledge.
Hassan, R. (2003). Network time and the new knowledge epoch. *Time and Society, 12*, 225–240.
Hassan, R., & Purser, R. E. (Eds.). (2007). *24/7: Time and temporality in the network society*. Stanford, Calif: Stanford Business Books.
Hatch, M. J., Kostera, M., & Kozminski, A. K. (2005). *The three faces of leadership: manager, artist, priest*. Malden, MA: Blackwell Pub.
Hertzum, M. (2002). The importance of trust in software engineers' assessment and choice of information sources. *Information and Organization, 12*, 1–12.
Hippel, E. v. (1988). *The sources of innovation*. New York: Oxford University Press.
Hochschild, A. R. (1983). *The managed heart: Commercialization of human feeling*. Berkeley: University of California Press.
Hochschild, A. R. (1997). *The time bind: When work becomes home and home becomes work*. New York: Metropolitan Books.
Höpfl, H. (1995). Rhetoric and the threat of ambivalence. *Studies in Cultures Organizations and Societies, 1*(2), 175–187.
Howcroft, D., & Wilson, M. (2003). Paradoxes of participatory practices: The Janus role of the systems developer. *Information and Organization, 13*(1), 1–24.
Humphrey, W. S. (1997). *Managing technical people: Innovation, teamwork, and the software process*. Reading, Mass: Addison-Wesley.
Jacobs, J. A., & Gerson, K. (2004). *The time divide: Work, family, and gender inequality*. Cambridge, Mass: Harvard University Press.
James, G. (1986). *The tao of programming*. Santa Monica: Infobooks.

Jemielniak, D. (2007). Managers as lazy, stupid careerists? Contestation and stereotypes among software engineers. *Journal of Organizational Change Management, 20*(4), 491–508.
Jemielniak, D. (2008). Engineers or artists–programmers' identity choices. *Tamara Journal of Critical Organization Inquiry, 7*(1), 20–36.
Kanter, R. M. (1977). *Men and women of the corporation*. New York: Basic Books.
Keil, M., & Robey, D. (1999). Turning around troubled software projects: An exploratory study of the deescalation of commitment to failing courses of action. *Journal of Management Information Systems, 15*(4), 63–87.
Kemerer, C. F. (1987). An empirical validation of software cost estimation models. *Communications of the ACM, 30*(5), 416–429.
Kesteloot, L. (2003). Why software is late. <http://www.teamten.com/lawrence/writings/late_software.html>. Retrieved accessed 12.12.07.
Kidder, T. (1981). *The soul of a new machine*. Boston: Little – Brown.
Klein, H. K., & Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quarterly, 23*(1), 67–94.
Kleine, M. (1990). Beyond triangulation: ethnography, writing, and rhetoric. *Journal of Advanced Composition, 10*(1), 117–125.
Knights, D., & Willmott, H. (1999). *Management lives: Power and identity in work organizations*. London, Thousand Oaks, Calif: Sage.
Kociatkiewicz, J., & Kostera, M. (2003). Shadows of silence. *Ephemera, 4*(3), 305–313.
Kostera, M. (2007). *Organizational ethnography. Methods and inspirations*. Lund: Studentlitteratur.
Kraft, P. (1977). *Programmers and managers: The routinization of computer programming in the United States*. New York: Springer-Verlag.
Kraut, R. E., & Streeter, L. A. (1995). Coordination in software development. *Communications of the ACM, 38*(3), 69–81.
Kunda, G. (1992). *Engineering culture: Control and commitment in a high-tech corporation* (Rev. ed.). Philadelphia, PA: Temple University Press.
Kunda, G., & Van Maanen, J. (1999). Changing scripts at work: Managers and professionals. *Annals of the American Academy of Political & Social Science, 561*(1), 64–80.
Latour, B. (1986). The powers of association. In J. Law (Ed.), *Power, action and belief – A new sociology of knowledge?* London, Boston, Henley: Routledge & Kegan Paul.
Latusek, D., & Jemielniak, D. (2007). Trust in software projects: Thrice told tale. *The International Journal of Technology, Knowledge and Society, 3*(10), 117–125.
Leidner, R. (1993). *Fast food, fast talk: service work and the routinization of everyday life*. Berkeley: University of California Press.
Macrae, N. C., Stangor, C., & Hewstones, M. (Eds.). (1996). *Stereotypes and stereotyping*. New York: Guilford Press.
Mallet, S. (1975). *Essays on the new working class*. St. Louis: Telos Press.
Nandhakumar, J. (2002). Managing time in a software factory: Temporal and spatial organization of IS development activities. *The Information Society, 18*, 251–262.
Negroponte, N. (1996). *Being digital*. New York: Vintage Books.
O'Carroll, A. (2008). Fuzzy holes and intangible time. Time in a knowledge industry. *Time and Society, 17*, 179–193.
O'Conaill, B., & Frohlich, D. (1995). *Timespace in the Workplace: Dealing with Interruptions.* Paper presented at the conference on human factors in computing systems.
Orlikowski, W. J., & Yates, J. (1999). It's about time: Temporal structuring in organizations. *Organization Science, 13*(6), 684–700.
Orr, J. E. (1996). *Talking about machines: An ethnography of a modern job*. Ithaca, NY: ILR Press.
Perlow, L. A. (1997). *Finding time: How corporations, individuals, and families can benefit from new work practices*. Ithaca, NY: ILR Press.
Perlow, L. A. (1998). Boundary control: The social ordering of work and family time in a high-tech corporation. *Administrative Science Quarterly, 43*(2), 328–357.
Perlow, L. A. (1999). The time famine: Towards a sociology of work time. *Administrative Science Quarterly, 44*, 57–81.
Perlow, L. A. (2001). Time to coordinate: Toward an understanding of work–time standards and norms in a multicountry study of software engineers. *Work and Occupations, 28*(1), 91–111.
Perlow, L. A. (2003). *When you say yes but mean no: How silencing conflict wrecks relationships and companies and what you can do about it* (1st ed.). New York: Crown Business.
Perry, D. E., Staudenmayer, N. A., & Votta, L. G. (1994). People, organizations, and process improvement. *Software, IEEE, 11*(4), 36–45.
Pollack, J. (2007). The changing paradigms of project management. *International Journal of Project Management, 25*(3), 266–274.
Rappaport, R. A. (1992). Ritual, time, and enternity. *Zygon Journal of Religion and Science, 27*(1), 5–30.
Reel, J. S. (1999). Critical success factors in software projects. *IEEE Software*(May/June), 18–23.
Reich, R. B. (2000). *The future of success*. New York: A. Knopf.
Rennecker, J., & Godwin, L. (2005). Delays and interruptions: A self-perpetuating paradox of communication technology use. *Information and Organization, 15*(3), 247–266.
Robinson, J. P., & Godbey, G. (1999). *Time for life: The surprising ways Americans use their time*. University Park: Pennsylvania State University Press.
Sabelis, I. (2001). Time management. Paradoxes and patterns. *Time and Society, 10*(2/3), 387–400.
Sahay, S. (1997). Implementation of information technology: A time–space perspective. *Organization Studies, 18*(2), 229–260.
Schor, J. (1991). *The overworked American: the unexpected decline of leisure*. New York: Basic Books.
Schwartzman, H. (1993). *Ethnography in organizations*. Newbury Park, London, New Delhi: Sage.
Senge, P. M. (1990). *The fifth discipline: The art and practice of the learning organization*. New York: Doubleday Currency.
Shih, J. (2004). Project time in silicon valley. *Qualitative Sociology, 17*(2), 223–245.
Smith, J. H., & Keil, M. (2003). The reluctance to report bad news on troubled software projects: A theoretical model. *Info Systems Journal, 13*, 69–95.
Stewart, T. A. (1997). *Intellectual capital: The new wealth of organizations*. New York: Doubleday/Currency.

Styhre, A., & Sundgren, M. (2005). *Managing creativity in organizations: Critique and practices*. Basingstoke, New York: Palgrave Macmillan.
Thompson, E. (1967). Time, work discipline and industrial capitalism. *Past, Present and Future, 38*, 56–97.
Trice, H. M., & Beyer, J. M. (1993). *The cultures of work organizations*. Englewood Cliffs, NJ: Prentice Hall.
Ullman, E. (1995). Out of time: Reflections on the programming life. In J. Brook & I. Boal (Eds.), *Resisting the virtual life: The culture and politics of information*. San Francisco – Monroe: City Lights Publishers.
Vallas, S. P. (2003). The adventures of managerial hegemony: Teamwork, ideology, and worker resistance. *Social Problems, 50*(2), 204–225.
Van de Ven, A. H., & Polley, D. (1992). Learning while innovating. *Organization Science, 3*, 92–116.
Van Maanen, J. (1988). *Tales of the field: On writing ethnography*. Chicago: University of Chicago Press.
Walsham, G. (1993). *Interpreting information systems in organizations*. New York: Wiley.
Weick, K. E. (1969). *The social psychology of organizing*. Reading, Mass: Addison-Wesley Pub. Co.
Westenholz, A. (2006). Identity, times and work. *Time and Society, 15*(1), 33–55.
Whyte, W. F., & Whyte, K. K. (1984). *Learning from the field: A guide from experience*. Beverly Hills: Sage Publications.
Zerubavel, E. (1979). *Patterns of time in hospital life: A sociological perspective*. Chicago: University of Chicago Press.
Zerubavel, E. (1981). *Hidden rhythms: Schedules and calendars in social life*. Chicago: University of Chicago Press.
Zuboff, S. (1988). *In the age of the smart machine: The future of work and power*. New York: Basic Books.