TAMARA Journal Vol 7 Issue 7.1 2008 ISSN 1532-5555

Software engineers or artists? Programmers' identity choices

Dariusz Jemielniak
Kozminski Business School, Poland

A novice programmer was once assigned to code a simple financial package. The novice worked furiously for many days, but when his master reviewed his program, he discovered that it contained a screen editor, a set of generalized graphics routines, an artificial intelligence interface, but not the slightest mention of anything financial. When the master asked about this, the novice became indignant. "Don't be so impatient," he said. "I'll put in the financial stuff eventually." (James, 1986)

ABSTRACT

This paper explores how assigning software developers the identity of "engineers" metes out specific assumptions about IT projects. To this end, the paper describes an alternative metaphor of programming as art, which is commonly used by the programmers interviewed. In addition, the discussion draws conclusions from the discrepancies between the two views as well as from the proposed metaphor, explaining organizational reluctance to aesthetical vocabulary. This paper discusses occupational identity—emphasizing the identity of programmers—using qualitative research methods. As such, it enriches the literature currently available on this profession.

INTRODUCTION

Software development has traditionally been perceived as an engineering field (McDermid, 1991; Pressman, 1992; Lewerentz and Rust, 2000), which is even reflected in this field's common name: "software engineering." Initially, as in all young disciplines, software development was defined by categories and meanings taken from others (Bryant, 2000). This process was by no means unusual; for example, management theory was also initially understood as an engineering subdivision, and this approach prevails even today in some business concepts (Boje and Winsor, 1993; Shenhav, 1999). Moreover, comparing software creation to the formation of material construction is certainly a metaphor that is useful in many respects. However, as with any metaphor, accentuating certain traits of the described phenomenon obscures others. Although plausible, it also carries a risk of becoming accepted as the only possible view—a cognitive mistake that narrows our minds. Indeed, as long as we do not perceive that we deal with a theoretical abstract rather than with "reality itself," we tend to ignore anything that does not fit the image (Morgan, 1983, 1986). However, it should not be forgotten that, although commonly described in a similar language, software is by no means a physical construction. It may resemble such in many respects, yet it does not in others.

As Anthony Bryant (2000) points out, the current widespread and unreflective acceptance of this metaphor may have a huge negative impact on the practice of programming. Indeed, the way in which software developers' professional identity is defined may influence IT businesses in many respects. Perceiving programmers[2] as engineers easily leads to giving them "construction specifications" instead of joint brainstorming on what the software can do (as would be the case, for example, with interior designers). Such an approach results in perceiving the knowledge about software as

[2] In this paper I intentionally use the word *programmer* instead of *software engineer*, although in the common language they are not entirely interchangeable and a programmer is sometimes considered to have a lower labor status.

quantitative, clearly structured, and orderly rather than qualitative, based on intuition, and messy. Thus, programmers begin to be seen as interchangeable agents who can take over work from each other or whose efficiency can be improved by assigning more people to the project. These blatant assumptions (Brookes, 1995) may all play a role in constructing the role of a software creator as an engineer. They may also have another negative side effect—namely, programming is reduced to a mechanical task that, although requiring a lot of intelligence, is rather individual than collective. In reality, the social side of software development is quite crucial (Kociatkiewicz and Kostera, 2003).

In this light, it should not be surprising that programmers quite often do not characterize themselves as engineers. In fact, they criticize such bracketing (Brookes, 1995; Beizer, 2000; McBreen, 2002). However, the alternatives for engineer identity representation are much less described. Therefore, it may be interesting to search for other useful metaphors of software development and determine other ways in which programmers define their own vocational identity.

IDENTITY BATTLEFIELD

The way in which a given occupation is addressed is by no means trivial. The imposed identity implicates a whole set of behaviors and assumptions. The enactment of occupational identities is especially interesting in times of liquid careers and roles (Bauman, 2007). Contemporary organizations are characterized by more individualized and short-term organizational identities, in decline of stable organizational structure and career-paths (Bauman, 1998; Beck, 2000). Thus, while posts, responsibilities, employers, and even the very content of work do change, especially in case of knowledge-workers the professional identity stays. The way key organizational actors are perceived and defined is, therefore, of utter importance.

For the purpose of this paper, an (occupational) identity is understood as a combination of social roles and expectations, enacted jointly by the actor, the organizational script, and the surrounding environment and resulting in a consistent collection of assumptions toward the view of self (Ashforth and Kreiner, 1999). Occupational self-definition (an identity) is a means of situating the carrier in a wider context and determining what s/he does, believes, or even feels (Ashforth and Humphrey, 1995); it determines to a large extent how people are perceived and is one of the key components of social power. This practice of continuously identifying one's self and others, as well as continuously receiving and applying labels, is inevitably embedded in organizational discourse (Grant, Keenoy and Oswick, 1998; Knights and Willmott, 1999). In this sense, it is useful to perceive organizational identities as "socially constructed stories about individuals and their surroundings as they engage in social work practices" (Westenholz, 2006: 1018).

Michael Rosen writes about managerial manipulation at an advertising agency in such words (1985/91: 89):

> Managerial domination in practice is maintained not by an excited audience rushing back to the agency to work energetically, but by a workforce accepting the defined terrain. Here culture, creating and being the terrain for consciousness, is a mechanism for control.

So is the identity of the occupation, one may add.

These organizational identities are constructed in a melting pot of one's own and imposed images. They are created in a never-ending process of grinding, blending, and forming the perceptions and demands received from the others and simultaneously the views offered to them for consideration. In this sense, it may be more useful to speak of the ongoing process of obtaining an identity rather than of "having one" (Weick, 1969/79; Sveningsson and Alvesson, 2003).

This identity tension is intertwined with the fundamental conflict of an individual with an organization (and thus between an employee and a manager, a personification of the administrative power). Zygmunt Bauman (1992/98: 98) even asserts that individuality is inevitably the primary enemy for any organization. To control its members, the organization has to deprive them of their uniqueness, bracket them, and impose an identity as standardized as possible. This process is even more common in knowledge-intensive[3] companies.

Such corporations typically rely heavily on the normative/ideological control over their employees (Kunda, 1992; Hochshild, 1997). Although in manual labor the strict observation of physical movements in Tayloristic manner suffices to achieve satisfactory results (a person is but a substitute for a machine), in the case of knowledge-work it, by definition, cannot be successful. Therefore, in this case, the disciplinary power of the organization over its employees goes beyond just the body to rely on their internalizing the external rules and, finally, on self-control (Foucault, 1977). Consequently, personal surveillance is much less common than the control of outcomes or even just the performance (Ouchi and Maguire, 1975; Meiksins and Watson, 1989). Thus, the companies supplement the control of bodies with the control of what happens in the minds of the workers. Paradoxically, in this sense, “blue collar” workers may have more freedom of thought and more intact integrity than “white collar” workers (Leidner, 1993). Where behavior cannot be bracketed and prescribed by procedure or direct supervision, the self-definition and devotion of the worker play crucial role. In fact, as Rosabeth Moss Kanter writes (1977: 65):

> …the organization’s demands for a diffuse commitment (…) is another way to find concrete measures of trust, loyalty, and performance in the face of uncertainties

The meaning of managerial work shifts accordingly, evolving from the traditional industrially bureaucratic model, in which the standardization of work process, planning, structural design, control, and formalization are most important (Mintzberg, 1983). As a result, identity shaping, indoctrination, and the creation of emotions become parts of managers’ routines—or at least intentions (Jackall, 1988; Kärreman and Alvesson, 2004). Consequently, the process of the employees themselves creating occupational identity and the resistance to accepting managerial bracketing of a given profession are utterly important for understanding contemporary organizations. As such, the purpose of this paper is to present the alternative perceptions of software engineers’ identities, as described in open, unstructured qualitative interviews.

METHODOLOGY

In order to understand the programmers’ other self-identity manifestations, the researcher conducted a series of open-ended, unstructured interviews, lasting typically 40 to 50 minutes, in 3 Polish and 2 American IT companies; 56 software engineers and 4 managers participated in the interviews. The interviews were conducted in a free-flow manner, following the stories described by the interviewees in an effort to understand their perceptions of their own occupations and work. As such, the structure of the interviews was very much dependent on the interviewees (Whyte, 1984). The studied companies were of different sizes, but all developed software for the corporate market (banking, aviation, and customer service solutions).

In the interviews, particular emphasis was put on storytelling and the narrative aspects of organizational life (Boje, 2001),

[3] The strict definition of knowledge-intensive work is difficult, if not nearly impossible, to identify. At the general level, it refers to jobs requiring mostly intellectual skills of highly educated employees. However, knowledge is a concept that is unambiguously positive, yet difficult to measure. For example, it is unclear how to decide whether a plumber has to know less than a surgeon or not. Thus, it should be stressed that knowledge intensiveness is a social construct (Alvesson, 2000; Styhre and Sundgren, 2005), just like “professional” status (Brante, 1988).

which possibly influenced the methods of research, even if not evident in the particular outcomes of the study. In all cases, the interviews were conducted on site, usually during lunch breaks; all were recorded and transcribed, with the interviewees' consent. Throughout the process of interviewing, a working model for understanding the important cognitive categories was being developed; consequently, the later interviews were increasingly grounded in the issues covered by earlier ones. In three cases, the initial interviews were followed by additional meetings.

In this sense, the study was qualitative and grounded-theory inspired (Glaser and Strauss, 1957). It was a preliminary part of a project subsequently conducted as an anthropological field study (Czarniawska-Joerges, 1992; Schwartzman; 1993). Thus, it serves as an introductory part to an organizational ethnography study (Kostera, 2007).

It should be also noted that the study was performative, not ostensive (Latour, 1986). As such, it aimed to understand and explain the point of view of the interviewed, rather than offer an ultimate interpretation of the analyzed problem "in reality." Undoubtedly, the presented opinions are only one of many available perspectives and in no way more valid than those of other organizational groups. Still, learning how the software creators see their work is very important for understanding work in IT companies better or for explaining the tension between programmers and managers, which is quite typical for many corporations in this industry (Kidder, 1981; Kunda, 1992). After all, apart from "how it really is" (which may be extremely difficult to determine), it is also very important to realize how a crucial group of people engaged in a project understands and describes their own tasks and identities.

PROGRAMMING AS ART

The image that emerged from the interviews was particularly interesting in that software writers quite often described their work using the vocabulary and metaphors of art, not of engineering. As developing software is essentially based on writing and creativity is a crucial factor in software preparation, such an outcome is quite understandable. For many, it is worth viewing programming in terms of art (Knuth, 1974; Ditlea, 1984; Lyman, 1995; Ullman, 1996; Humphrey, 2000; Cox et al., 2001, 2004; Piñeiro, 2003; Bond, 2005). Many scholars have also stated that organization theory may benefit significantly from artistic inspirations (e.g., Adorno, 1973; Guillet de Monthoux and Czarniawska-Joerges, 1994; Höpfl, 1995; Kostera, 1997; Hatch, Kostera and Kozminski, 2005); indeed, in recent years growing interest has emerged in studies of aesthetical aspects of organizing (Strati, 1999; Linstead and Höpfl; 2000; Carr and Hancock, 2002). However, many of the modern organizations still ignore and neglect this way of defining programmers' selfhood.

In the case of software development, the perception of programmers' identity as defined through art is especially interesting because the view expressed by the programmers themselves is quite often very different from the common idea on what software development is about, and, consequently, from the prevailing metaphor used for describing programming in organizations. Therefore, the current paper depicts this alternative view on programming in the words of the informants[4] in order to demonstrate how software creation can be perceived as art. The paper then exploits this alternative metaphor and compares software writing to art. Finally, it examines the reasons for the tension between the widely accepted identity of a programmer as an engineer and the perception of the occupation as artistic in the eyes of software creators themselves, drawing upon the occupational identity theory—introduced at the beginning of this work—to show that the label of an engineer successfully imposed on programmers helps

[4] This word is most unfortunate for describing people and may carry a connotation of condescension. I use it merely to alternate between terms.

managers in sustaining their dominance and control.

WHY ART VERSUS ENGINEERING?

The use of the “programming as art” metaphor in this paper is based on four major ideas:

- programming is treated as an artistic hobby by programming professionals (engaging in an activity as both a hobby and professionally is generally unusual, but quite typical in artistic vocations);
- according to interviewees, artistic creativity is crucial in their job;
- interviewees often compared programming to art; and
- interviewees used aesthetical terms to describe their work.

“Art” in this article is understood similarly as in the approach of Harold Osborne, who claimed that “whatever among artifacts is capable of arousing and sustaining aesthetic experience in suitably prepared subjects we call a work of art” (1981:3). In this sense, programming is art whenever it evokes aesthetical feelings (and requires creativity, at least in some of its aspects). Such a view allows for the exploitation of the metaphor more effectively in occupations traditionally not perceived as artistic than the classical definition by Dickie (1974), which states that art is whatever the artistic world labels as such. For other approaches to this issues, see for example Dean (2003).

RESULTS

The first aspect related to programming that was identified as unusual and made the perception of software creation in terms of art more viable was hobbyist programming. Most of the informants complained that they spend a lot of time at work. More surprisingly, a significant number of them did some programming at home too. The following excerpt from one interview is particularly representative (Minicorp4):

[Q:] Do you ever think about problems from work once you go home?

[A:] Well, it depends. It all depends on what I do. I prefer to do some hobbyist stuff, so I write at home quite often.

[Q:] Are these programs that you use back at work or something else?

[A:] No. These are totally different programs, written just for fun.

[Q:] So you mean you work your usual hours in the company and then go home and additionally write other programs?

[A:] Well, I usually spend a couple of hours per week, which is why such homemade projects take a couple of months…These are not important things.

[Q:] So why do you do them?

[A:] You know, many people have similar side projects…Especially if you have a look at how the open source environment is developing. These people have to write their programs at some time…I myself do so occasionally, small things. I don’t think these programmers spend their working time at this. Well, some of them are students…But most spend their time at work, and once they get home, they develop their own stuff. At least, that’s what I do.

[Q:] But why do you do that?

[A:] It is a hobby. Some people read books, some like to write poetry, and some like to write programs.

Although the interlocutor described his work as “unimportant,” he also said that he regularly spends time on it away from the office. Interestingly, he also compared programming to poetry.

Many other interviewees made similar comments, emphasizing the additional fun they had from such work, which was unattainable in their regular job (Wodan6):

[Q:] Do you work at home?

[A:] At home? No, definitely not. I mean, I do some hobbyist

> programming, but it is something totally else. (…) I move away from commercial and professional programming, and at home I can write, hmm, let's say a script that generates nice color pictures. And everybody knows you can't do this at work, nobody really needs it, but such amateur projects give you a sense of satisfaction, that you are doing something interesting.

According to this informant, he was developing programs for the mere satisfaction of creation and the fun of doing something interesting. He also made a clear distinction between what he did at the company and at home; the second was described as entertaining (as we may assume, in opposition to the job). This division, to some extent, may personalize the two iconic images of an engineer (patiently performing the mundane tasks in a cubicle) and an artist (spending hours, captivated by something nobody really needs, but giving satisfaction). However, only the former is associated with occupational identity and regarded as work. The same view was expressed by many other interlocutors and is in clear accord with art(ist) studies. For example, Howard Singerman (1999) postulates that the role of an artist is in direct conflict with predictability and standardization, especially in organizations expecting uniformity.

It may be worth noting here that the main difference between a "geek" (programmer) and a "normal" person, according to some of the programmers themselves, relies mainly on a passion for learning, an internal drive to understand things, and creativity. As one of the disputants at **slashdot.org**[5], the cult portal of software people, said:

> There will always be exceptional people in any field—Donald Knuth is one of the great computing minds of our time, but there are plenty of others as well. The truth of the matter is that the people who have a passion for learning and exploring computers will always be different, because while normal people are content to sit around and type in Microsoft Word, these people actually want to understand what is happening behind the screen and why exactly these things happen the way that they do. There's nothing wrong with the content people, but because they're content, their time and energy will be spent elsewhere in other passions.

Such an attitude was surprisingly widespread among the interviewees, most of whom completed programming projects in addition to their regular appointments. They also emphasized the role of creativity in writing software.

In fact, creativity was a treat most commonly referred to when describing the characteristics of a good programmer (Wodan7):

> [Q:] What distinguishes programming from other occupations?
> [A:] Well, many things, that's for sure…I have always thought that it is quite creative. And you can see the results of your creativity pretty fast. A good field…Well, there are software engineers who shuffle tons of code over the Internet, change one thing, get to know one program. And this supposedly proves they're creative. But there are also really talented people who write such things…masterpieces.

This interlocutor strongly emphasized the role of creativity in software creation, clearly stating that skillful programming requires talent and that excellent programs are "masterpieces."

Other interviewees also often described programs as artistic creations. As stated earlier, a number of them explicitly compared software to poetry (Sand9):

> [Q:] What would you compare programming to?

[5] http://slashdot.org/comments.pl?sid=92986&cid=7994621

[A:] I'll put it this way. Once I spoke to a mathematician. I was still back in high school, and we shared views on poetry and on new algorithms in math. Then he said that he personally didn't like poetry, but a new algorithm is something of this sort…
[Q:] Do you share this view?
[A:] Yes.
[Q:] Why?
[A:] Well, there are many similarities, for example inspiration. I think that most programming work doesn't require inspiration. It is mostly mundane, but you need this inspiration for some fragments, [some] pearls.

A program itself is a piece of writing in a particular language; it should not be surprising then that—for those whose command of the language is highest—some work is appealing for the beauty of its language, not just its functionality (Bond, 2005). Indeed, the majority of interviewees referred to aesthetic, non-functional criteria for program evaluation (Optirec3):

[Q:] Could you describe the criteria for software evaluation?
[A:] Well, there is an element of functionality, and some component of art…
[Q:] What do you mean by "component of art"?
[A:] Well, it is quite unobvious…It is something that can be appreciated only by people skilled in the same art. It's like…the whole program looks nice, but a programmer can tell it only if he sees it from the inside.
[Q:] Does the component of art matter then?
[A:] Well, in business practice it may in some particular situations, but normally efficiency is more important than being a piece of art. (…) The fact that programming is art for me is that in some cases it gives you pure pleasure.
[Q:] When does that happen?
[A:] Well, mostly when you program something that interests you, not something that is just a task to be finished. When you have some given tasks, you have to compromise. When you prepare a piece of art, it usually is not compromised.

In this example, the interviewee directly described programming as art. He further added that the criteria for understanding and evaluating it depend on the competence of the evaluator (quite a typical approach in "regular" art as well).

According to many of the informants, the beauty of the code does not have much to do with its functionality (Sand14):

[Q:] Do you have any paragons in programming?
[A:] Meaning a person or a group? Well, Russians create a beautiful code—at least they did some time ago; we were programming in quantities, they went for the quality. A person creates beautiful code when he wants to show "I can do it, too!" So he writes a nice program, distributes it as an open source project on the Internet, and people say "hey, that's a smart guy to do something like that."

Clearly, the aesthetics of the code do matter.

Some programmers also made the point that, even while programming commercially, they add functions that were not required by the client just because they considered them "interesting" or "potentially useful" just for the pleasure of "creating a nice function" (Wodan10):

[Q:] What characterizes good software?
[A:] Well, it depends upon the programmer. (...) There are people who can write good code but at the same time they can do it quickly—that doesn't mean they do it wrong, they can just

> restrict themselves. Because, you know, as in any occupation, you have temptations. I wrote a nice function here, maybe I'll develop it and use it at some later time....

Performing just for the sake of elegance fits well into the role of artist, although within the identity of an engineer this could be considered unprofessional. The same approach was described by Joe Schofield (2003: 83-84):

> Some software engineers may dabble outside the scope of the project if it interests them. "Out-of-the-box" software engineers are sometimes the source of creeping requirements and technology churn. I once observed a manager accuse a team of expanding the scope of the project without the customer's input. The team, he suggested, was improvising by adding business rules to the process model. Indeed, the team had been reporting progress, just not the progress expected under the project plan.

A number of the informants described metaphysical, inspiration-like experiences they had while programming (Minicorp3):

> [Q:] Does programming involve any emotions?
> [A:] Well, there are these moments…moments of nearly mystical enlightenment. I'm wondering to what extent they are a result of—I don't know—brain chemistry or just a prolonged sensor deprivation. But there are really such moments, when you see…I had a couple of these moments, even more often… Well, maybe it was influenced by how long I programmed or how deprived of stimuli I was or how much coffee I drank, but I enter this trance [and] the code bursts from my fingers. Fountains of code burst from my fingers, and everything works straightaway. It is difficult to achieve in normal office work, when somebody comes in and out and disturbs you and new demands appear…

Indeed, six interviewees explicitly presented the view that programming is art (Sand7):

> [Q:] What is programming most like, if you were to categorize it?
> [A:] Well, it is like…essentially it is mostly like art. We are creating software, and programming languages are sort of tools.

In this light, it is by no means surprising that the interviewees used artistic vocabulary and comparisons so often. Numerous examples of similar perceptions exist in literature. As early as 1974, Donald Knuth wrote an article with the symptomatic title "Computer programming as an art" in which he supported the exact same view: Programming is an artistic endeavor. Ullman, (1996) explains why:

> People imagine that programming is logical, a process like fixing a clock. Nothing could be further from the truth. Programming is more like an illness, a fever, an obsession. It's like those dreams in which you have an exam but you remember you haven't attended the course. It's like riding a train and never being able to get off.

Peter Case and Erik Piñeiro demonstrated (2006) that programmers' communities commonly use aesthetical criteria in their approaches to software. In fact, Piñeiro's (2003) most interesting book shows how programmers rely to huge extent on aesthetical evaluation of their creations. In Peter Lyman's study (1995) programmers who were also musicians considered playing an instrument as a metaphor of programming. A similar view is shared by Steve Ditlea (1984):

> Software encourages alternative thought processes: among the most successful of today's programmers

are musicians, night owls and free spirits. Imagination is at a premium. The soft culture beckons. We are entering an era of techno-romantics, children of the information revolution who are equally comfortable with the abstractions of technology and the emotions of the heart. (…) It may seem a contradiction to talk about techno-romantics: how can the precision of computer technology coexist with the whims of the human heart? Yet the entire history of computing is filled with pioneer techno-romantics, equally comfortable with the most fundamental secrets of logic and the universe of emotion.

According to Peter McBreen (2002), the standardization of software and the act of treating it as any other engineering creation lead to serious misconceptions of programming. He believes that programming can be taught mainly by collaborating with “masters” as it is strongly based on a tacit knowledge. In this respect, programming is closer to craftsmanship or art than to engineering. Some authors even take it further (Weber, 2004: 59):

> The essence of software design, like writing of poetry, is a creative process. The role of technology and organization is to liberate that creativity to the greatest extent possible and to facilitate its translation into working code. Neither new technology nor a “better” division of labor can replace the creative essence that drives the project.

All things considered, it is clear that, for some—if not the majority—of the programmers, software writing is defined at least as much within the concept of art as within the idea of engineering. For programmers themselves, it is a useful metaphor, present in their regular discourse. Although it is not the intention of this paper to prove the engineering metaphor to be less valuable than that of art as both have pros and cons, clearly one of the ways to identify programming is much less frequently used in official organizational language—despite its usefulness for the programmers themselves.

Yet, surprisingly, in all studied companies, the programmers’ posts were described as engineering. Moreover, during observations conducted on site, managers often used the metaphor of an engineer to address programmers. For example, the Wodan manager, when introducing the researcher to the company, stated:

> Our company has one of the finest collections of engineers in the country. If we can’t develop a piece of software you want, the chances are, nobody can.

In addition to the typical managerial chatter and boasting, he referred to his subordinates as engineers. In two cases, software development teams were also called “software engineers” in organizational documents. Furthermore, most of the interviewees had the word *engineer* in one form or another on their business cards.

Engineering was presented as something positive (Sandcorp manager):

> What we do is more problem solving than just software development. We try to address the customer’s needs even beyond what they initially think. Also, we always try to make our product as structured and logically created as possible. We do really good engineering, not a patchwork here and there.

On one occasion a manager explicitly contrasted a creative and engineering approach to programming (MinicorpM):

> You know, I don’t really want to have people who go into a creative trance and come out with a piece of excellent code nobody else can understand. It may be good by itself, but that’s not the point. We need teamwork, and a good software engineer is somebody who

> not only can write well, you know, but also write it in such a way that others can pick it up in the meantime, relate to it, know what's going on. I've seen a couple of "creative" [quotation marks gestured] guys who really didn't do that well on a team. It just doesn't work like that.

Thus, the managerial and organizational approach is quite clear. Programmers are perceived and referred to as engineers; they are expected to believe they are engineers. Not only is their role at stake (the expectation toward behavior), but also their imposed identity (the expectation toward self-perception). This discrepancy is interesting and worth considering.

CONCLUSIONS: CONTEMPORARY AVANT-GARDE

The evaluation of an employee's work becomes more dependent on his/her intensions and loyalty than just the result. In programming, where precise planning and an estimation of tasks' difficulty level are still developing, this increasing dependence is particularly visible. The positive evaluation of an employee is often highly correlated to time spent at work and other manifestations of dedication, such as coming back from vacation in the case of a crisis or putting family life second (Perlow, 1997, 2004; Cooper, 2000; Jemielniak, 2007). Such a correlation is one of the reasons for the advancing bifurcation of working time (Jacobs and Gerson, 2001) as employees increasingly face the choice of making a symbolical sacrifice and working very long hours or not working at all—at best working part-time or for hourly wages. Although such a situation agrees with the image of an educated engineer, it does not really fit the identity of an artistic genius. Perhaps this is one of the reasons why organizations and managers find it much more convenient to see programmers as engineers, even though the programmers themselves are sometimes having difficult time being defined as such (Minicorp7):

> Some of them [managers] don't get it. You can write something average, or you can write a really beautiful code. And this is something really great, when you can do this. But some managers think like this: here is the specification, do what is required as fast as you can, and that's it. They say "you're an engineer, so just go and do some engineering". It doesn't work like that.

In addition, an engineer (as portrayed in the iconic Dilbert comic figure) is definitely much more prone to manipulation than the free-spirited, independent artist. As the ideological control in non-ideological organizations grows in popularity (Czarniawska, 1988), one of the crucial managerial functions is enacting employees' identity, allowing for easier control. By calling software creators engineers, organizations justify their inter-changeability and imply standardization of this occupation. Yet, as evident from the presented research, this process is not occurring without resistance; many programmers, when denied the possibility to be artists at work, enter this role after hours.

Heather Höpfl (1995) compares two views on the world of organizations, distinguishing them as rhetoric (persuasion, the method of convincing people to bend to the organizational expectations) and poetics (often ignored by managers, relying on creativity, being a spontaneous reaction of organizational actors). Rhetoric is the "propaganda," the way in which the identity label is attributed, while poetics is the reaction, the recoil of the labeled. This polyphony, evident in all organizations, is what managers tend to suppress (Boje, 1995).

This process of assigning names plays a major role in applying social control. For example, the seemingly indifferent change in names from "drunkards" to "alcoholics" resulted in redefining the problem of drinking from a moral issue to a medical issue (Brown, 1988). Something as simple as chefs' instructions convey powerful roles and may help in reinforcing the gender gap (Silva, 2000). Similarly, the name by which a given occupation is described has a significant impact on the employee as it both defines the terrain in a way in which the management finds it convenient and exploits the right to name things and people, thereby reasserting authority.

In this sense, the organizational rhetoric of programmers' identity has to reinforce the official image ("engineers") and refute the poetic alternative view on programming. The more challenging the alternative metaphor for programming is, the more it has to be ignored. Thus, programmers perform art, but can only enjoy it to the full capacity at home, as a hobby.

Keeping all possible dangers of overexploiting a metaphor in mind, one may draw another parallel. This situation is quite reminiscent of the futurist artistic movement in the beginning of the 20th century. Jochen Schulte-Sasse (1984), writing about the modernist movement in art, recalls Adorno's thought—namely, bourgeoisie aims at unification and the elimination of individualism. An individual is free, but denied the right (or the possibility) to be original. Only the art—particularly the avant-garde/bohemia—can resist this tendency[6].

Indeed, the avant-garde movement relied to a large extent on questioning the social norm, including the criteria of art. It promoted individualism in opposition to the mass society. For example, Renato Poggioli (1968) writes that the art of avant-garde relies on negation, refuting the bourgeois society's norms. Surely, this is a limited revolt in which anarchy remains under control (Poggioli, 1968: 107):

> …in an epoch or culture like ours the artist finds himself "on strike (*en greve*) against society." But, in order to strike, one has to be employed

Moreover, according to Terry Eagleton (2003: 39):

> Because subjects like literature and art history have no obvious material payoff, they tend to attract those who look askance at capitalist notions of utility. The idea of doing something purely for the delight of it has always rattled the grey-bearded guardians of the state. Sheer pointlessness is a deeply subversive affair.

Perhaps this is why programmers write their own software at home.

From the point of profitability, programming is a gesture of protest as well. Although the programmers' rebellion is tame, they can be appreciated as real artists outside of work. To some extent, they exclude themselves from the system, like the original *bladerunners*[7]. Therefore, the rhetoric of calling programmers "engineers" and imposing a professional identity on them serves yet another purpose: allowing managers to try to "civilize" the rebels with whom they have to cope.

The application of artistic metaphor to programming has some explanatory power even on the very shallow level of dress codes. Programmers are considered the worst dressed occupation of all industries (Hearn, 2005). However, casual dress—just like bohemian negligence—could also be an act of denouncing the form (in this case, the managerial uniform), resistance toward the

[6] Important differences exist between bohemian and avant-garde movements (the latter contested bohemians for being civilized rebels, protesting mainly through dress, presence in elite circles, and the lack of social program, in order to dilute the distinction between the recipient and the creator of art as well as art and everyday life; see Bürger, 1974/1984). However, in this extensive comparison of programmers to both of these movements, a common theme emerges: negative perception of the bourgeoisie lifestyle. This perception is used as a metaphor to accentuate certain features of the programmers' community, not to claim they are "just like" bohemians/avant-gardists.

[7] *Blade Runner* is a cult science fiction movie by Ridley Scott that is favored by many "geeks," but also serves as a symbol used by Deleuze and Guattari (1986) to describe a person partly outside the system but still able to be on good terms with all parts of the conflict by providing them all with arms. This role agrees with the identity of unusual nomadic freedom (Jemielniak and Jemielniak, 1999).

standardization, and bracketing (Kidder, 1981; Kawasaki, 1990).

In this sense, using the notion of engineering in the context of software engineers may in fact be a way of reinforcing managerial dominance. Philip Kraft (1977) demonstrates that, in some cases, managers use the engineering professionalization ethos to impose their own created standards of work on the programmers and increase reign over them. Denying the artistic role of the programmers and identifying them with standard-educated engineers further makes them believe that they are more easily replaceable. Leslie Perlow (1998) explains how managerial expectations of programmers and appeals to their professional code serve as methods of exploiting the workers and forcing them to work more. Indeed, as Mats Alvesson (2000) points out, occupational identity in the case of programmers may impose temporal expectations of staying longer hours. As such, the identity of an engineer really does not allow for institutional and structural independence, but it helps—among other things—keep the wages lower. For example, in Kraft's study, managers were able to persuade their subordinates, at least officially, that comparing salaries is "unprofessional." This also allows for distancing an artist from his/her work as writings in programming language are rarely signed. In other words, the importance of the individual is belittled. Identifying programming as a standardized task also helps place the responsibility for any schedule slips[8] on programmers. Artistic identity is ignored by the management as, next to creativity and talent, it implicates unpredictability and high individualism as well. The displays of the artistic "hacker ethic"—even if part of the occupational face-work ritual (Goffman, 1967)—have to be, sticking to Goffman's vocabulary (1963), stigmatized as not fitting the organizational expectations.

[8] Falling behind the schedule is extremely widespread in software creation. Only 26 percent of software projects are delivered on time and within budget (Smith and Keil, 2003).

All things considered, it is quite understandable why programmers' identity is torn between two—if not more—images. As uniformity and dedication to the organization go first, programmers' perception of their job in terms of art is questioned and suppressed by the view of engineering. Compared to other occupations, programmers oppose this bracketing quite fiercely; they can be truly labeled as contemporary avant-garde. Still, they are continually challenged with a powerful metaphor of engineers. Although such a view of programming carries many misleading connotations and leads to various misconceptions in business life as well, it has one undisputable advantage in that it supports managerial domination by associating programming with a standardized, teachable, and predictable activity that is easily planned. In other words, occupational identity serves as a tool for ideological control. The internalization of engineers' identity by the programmers simply makes managing them easier—even if this is not how they truly perceive their work.

REFERENCES

Adorno, Theodor W. (1973) *Philosophy of Modern Music* (trans. W. Blomster). New York: Seabury.

Alvesson, Mats (2000) Social Identity and the Problem of Loyalty in Knowledge-Intensive Companies, *Journal of Management Studies*, 37(8), 1101-1223

Ashforth, Blake E., and Humphrey, Ronald H. (1995) Labeling processes in the organization: Constructing the individual. In L. L. Cummings & B. M. Staw (eds.) *Research in organizational behavior*, vol. 17, pp. 413-461. Greenwich, CT: JAI Press.

Ashforth, Blake E., and Kreiner, Glen E. (1999) "How Can You Do It?": Dirty Work and the Challenge of Constructing a Positive Identity, *The Academy of*

Management Review, vol. 24(3), pp. 413-434.

Bauman, Zygmunt (1992/98) *Smierc i niesmiertelnosc – o wielosci strategii zycia (Mortality, Immortality and Other Life Strategies)*, Warszawa: PWN

Bauman, Zygmunt (1998) *Liquid modernity,* Cambridge: Polity Press

Bauman, Zygmunt (2007) *Liquid times: living in an age of uncertainty,* Cambridge: Polity Press

Beck, Ulrich (2000) *The brave new world of work,* Cambridge - Malden, Mass.: Polity Press.

Beizer, Boris (2000) 'Software is different' in: *Annals of Software Engineering* 10, 293-310

Boje, David M. (1995) Stories of the storytelling organization: A postmodern analysis of Disney as "Tamara-land." *Academy of Management Journal* 38(4): 997-1035.

Boje, David M. (2001) *Narrative Methods for Organizational and Communication Research*, London: Sage.

Boje, David M. and Winsor, Robert D. (1993) The Resurrection of Taylorism: Total Quality Management's Hidden Agenda, *Journal of Organizational Change Management*, no. 6, 57-70

Bond, Gregory W. (2005) 'Software as art' in: *Communications of the ACM*, vol. 48 no. 8, 118-124

Brante, Thomas (1988) Sociological Approaches to the Professions, *Acta Sociologica* 31(2), 119-142

Brookes, Frederick P. (1995) *The Mythical Man-Month,* Reading, Massachusetts: Addison- Wesley

Brown, Richard H. (ed.) (1998) *Toward a Democratic Science: Scientific Narration and Civic Communication*, Yale: Yale University Press

Bryant, Anthony (2000) Metaphor, myth and mimicry: The bases of software engineering in: *Annals of Software Engineering* 10, 273-292

Bürger, Peter (1974/1984) *Theory of the Avant-Garde,* Minneapolis: University of Minnesota Press

Carr, Adrian and Hancock, Philip (2002) Art and aesthetics at work: An overview, in *Tamara: Journal of Critical Postmodern Organization Science*, 2(1) 1-7 (introduction to the special issue on organizational aesthetics)

Case, Peter and Piñeiro, Erik (2006) Aesthetics, performativity and resistance in the narratives of a computer programming community, *Human Relations*, vol. 59(6), pp. 753-782

Cooper, Marianne (2000) Being the "Go-To Guy": Fatherhood, Masculinity, and the Organization of Work in Silicon Valley, *Qualitative Sociology* 23(4), 379-405

Cox, Geoff and Krysa, Joanna (2003) Art as Engineering: Techno-Art Collectives and Social Computer Change, *Art Inquiry*, Lodzkie Towarzystwo Naukowe, Lodz, available at: http://www.anti-thesis.net/texts/engineering.pdf

Cox, Geoff, Ward, Adrian and McLean, Alexander (2001) The Aesthetics of Generative Code, in E. Thacker (ed.) *Hard_Code: narrating the network society*, Alt-X Digital Publishing

Cox, Geoff, Ward, Adrian and McLean, Alexander (2004) Coding Praxis: reconsidering the aesthetics of code, paper prepared for the *Read_Me* conference, Aaahus, available at http://www.anti-thesis.net/texts/praxis.pdf

Czarniawska-Joerges, B. (1988) *Ideological control in nonideological organizations*, New York: Praeger

Czarniawska-Joerges, Barbara (1992) *Exploring complex organizations: A cultural perspective*, SAGE: Newbury Park – London – New Delhi

Czarniawska-Joerges, Barbara and Pierre Guillet de Monthoux (1994) Good novels, better management: Reading organizational realities. Harwood Academic Publishers

Dean, Jeffrey T. (2003) The Nature of Concepts and the Definition of Art, *The Journal of Aesthetics and Art Criticism*, vol. 61(1), pp. 29-35.

Deleuze, Gilles and Guattari, Felix (1986) *Nomadology: The War Machine (Traite de nomadologie: La machine de guerre)*, New York: Semiotext(e),

Dickie, George (1974) *Art and the Aesthetic. An Institutional Analysis,* Ithaca: Cornell University Press

Ditlea, Steve and Lunch Group (1984) 'Beautiful!' in: Digital Deli – The Comprehensive, User-Lovable Menu of *Lore, Culture, Lifestyles and Fancy*, New York: Workman Publishing Company, available also at http://www.atariarchives.org/deli/soft.php

Eagleton, Terry (2003) *After Theory*, New York: Basic Books

Foucault, Michel (1977) *Discipline and Punish. The Birth of the Prison*, London: Penguin Books

Foucault, Michel (1982) The subject and power, in: H. Dryfus and P. Rabinow, (Eds.) *Michel Foucault: Beyond Structuralism and Hermeneutics*. London: Harvester Wheatsheaf.

Glaser, Barney and Strauss, Anselm (1957) *Discovery of Grounded Theory: Strategies for Qualitative Research*, Chicago: Aldine

Goffman, Erving (1963) *Stigma,* Englewood Cliffs: Prentice-Hall

Goffman, Erving (1967) *Interaction Ritual: Essays on Face-to-Face Behavior*, New York: Random House, Inc.

Grant, David; Keenoy, Tom and Oswick, Cliff (eds.) (1998) *Discourse and Organization*, London – Thousand Oaks – New Delhi: SAGE

Guillet de Monthoux, Pierre and Czarniawska-Joerges, Barbara (eds.) (1994) *Good Novels, better management-reading organizational realities in fiction,* Chur: Harwood

Hatch, Mary Jo; Kostera, Monika and Kozminski, Andrzej K. (2005) *The three faces of leadership: Manager, artist, priest.* Blackwell Publishing

Hearn, Louisa (2005) IT workers dubbed 'worst dressed', *The Sydney Morning Herald*, 17 Nov. 2005, http://www.smh.com.au/articles/2005/11/17/1132016909640.html

Hochshild, Arlie R. (1997) *The time bind: When work becomes home and home becomes work,* New York: Metropolitan

Höpfl, Heather (1995) Organizational rhetoric and the threat of ambivalence, *Studies in Cultures, Organizations and Societies,* 1/2, 175-187

Humphrey, Watts S. (2000) Software – a performing science?, *Annals of Software Engineering* 10, 261-271

Jackall, Robert (1988) *Moral Mazes: The World of Corporate Managers*, New York: Oxford University Press

Jacobs, Jerry A. and Gerson, Kathleen (2001) Overworked Individuals or Overworked Families?, *Work and Occupations* vol. 28(1), pp. 40-63

James, Geoffrey (1986) *The Tao of Programming*, Infobooks, Santa Monica

Jemielniak, Dariusz (2007) Managers as lazy, stupid careerists? Contestation and stereotypes among software engineers, Journal of Organizational Change *Management*, vol. 20(4), pp. 491-508

Jemielniak, Dariusz and Jemielniak, Joanna (1999) Identity in a Time of Change, *Knowledge Transfer*, 2(1), 1-11

Kanter, Rosabeth M. (1977) *Men and Women of the Corporation,* New York: Basic Books

Kawasaki, Guy (1990) *The Macintosh Way*, Glenview: Foresman&Co.

Kärreman, Dan and Alvesson, Mats (2004) Cages in Tandem: Management Control, Social Identity, and Identification in a Knowledge-Intensive Firm, *Organization*, vol. 11(1), pp. 149-175

Kidder, Tracy (1981) *The Soul of A New Machine*, New York: Avon Books

Knights, David and Willmott, Hugh (1999) *Management Lives: Power and Identity in Work Organisations*, London, Sage

Knuth, Donald E. (1974) Computer programming as an art, *Communications of the ACM*, vol. 17(12), pp. 667-673

Kociatkiewicz, Jerzy and Kostera, Monika (2003) Shadows of Silence, *Ephemera* vol. 4/3, pp. 305-313

Kostera, Monika (1997) The Kitsch-Organization, *Studies in Cultures, Organizations and Societies* 3, 163-177

Kostera, Monika (2007) *Organizational ethnography. Methods and inspirations.* Lund: Studentlitteratur

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

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

Latour, Bruno (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, Dominika and Jemielniak, Dariusz (2007) (Dis)trust in Software Projects: A Thrice Told Tale. On Dynamic Relationships between Software Engineers, IT Project Managers, and Customers, *The International Journal of Technology, Knowledge and Society*, vol. 3(1), pp. 117-125

Leidner, Robin (1993) *Fast food, fast talk: Service work and the routinization of everyday life*, University of California Press: Berkeley-Los Angeles-London

Lewerentz, Klaus and Rust, Heinrich (2000) Are software engineers true engineers?, *Annals of Software Engineering* vol. 10, pp. 311-328

Linstead, Stephen and Höpfl, Heather (eds.) (2000) *The aesthetics of organization*, London, SAGE

Lyman, Peter (1995) Computing as Performance Art, *Edurom Review*, Jul/Aug, vol. 30 issue 4

McBreen, Pete (2002) *Software Craftsmanship: The New Imperative,* Boston-San Francisco-New York et al.: Addison-Wesley

McDermid, John A. (ed.) (1991), *The Software Engineer's Reference Book*, Butterworth Heinemann, London

Meiksins, Peter F. and Watson, James M. (1989) Professional autonomy and organizational constraint: The case of engineers, *Sociological Quarterly*, Winter, vol. 30 issue 4, pp: 561-585

Mintzberg, Henry (1983) *Structures in Fives: Designing Effective Organizations*, Englewood Cliffs NJ: Prentice Hall

Mintzberg, Henry (1998) Covert leadership: on managing professionals. Knowledge workers respond to inspiration, not supervision, *Harvard Business Review*, Nov-Dec, 140-147

Morgan, Gareth (1983) More on Metaphor: Why We Cannot Control Tropes in Administrative Science, *Administrative Science Quarterly* no. 28, 601-607

Morgan, Gareth (1986/1997) *Obrazy organizacji (Images of organization)*, Warszawa, PWN

Ouchi, William G. and Maguire, Mary Ann (1975) Organizational control: Two functions, *Administrative Science Quarterly,* no. 20, pp: 559-569

Perlow, Leslie A. (1997) *Finding Time. How Corporations, Individuals, and Families Can Benefit from New Work Practices*, Ithaca-London: ILR Press

Perlow, Leslie A. (1998) Boundary Control: The Social Ordering of Work and Family Time in a High-tech Corporation, *Administrative Science Quarterly* 43, 328-357

Perlow, Leslie A. (2004) *When you Say Yes But Mean No*, New York: Crown Business

Piñeiro, Erik (2003) *The Aesthetics of Code. On excellence in instrumental action*, Ph.D. dissertation available at http://www.lib.kth.se/Fulltext/pineiro031128.pdf

Poggioli, Renato (1968) *The Theory of the Avant-Garde*, Cambridge, Harvard University Press

Pressman, Roger S. (1992), *Software Engineering: A Practitioner's Approach*, 3rd edition, London, McGraw Hill

Rosen, Michael (1985/91) Breakfast at Spiro's: Dramaturgy and Dominance, in Frost, P. J., Moore, L. F., Louis, M. R., Lundberg, C. C. & Martin, J. (Eds.) *Reframing Organizational Culture*, ewbury Park - London - New Delphi: SAGE.

Schofield, Joe (2003) Observations from the ant hill: what ants and software engineers have in common, *Information Systems Management*, 20(1), 82-85

Schulte-Sasse, Jochen (1984) Foreword: Theory of Modernism versus Theory of the Avant-Garde, in: Bürger, P. (1974/1984) *Theory of the Avant-Garde,* Minneapolis: University of Minnesota Press

Schwartzman, Helen (1993) *Ethnography in organizations*, Sage: Newbury Park – London – New Delhi

Shenhav, Yehouda (1999) *Manufacturing Rationality: The Engineering Foundations of the Managerial Revolution*, Oxford: Oxford University Press

Silva, Elizabeth B. (2000) The cook, the cooker and the gendering of the kitchen, *The Sociological Review*, vol. 48, pp. 612-628

Singerman, Howard (1999) *Art Subjects: Making Artists in the American University*, University of California Press: Berkeley-Los Angeles-London

Smith, H. Jeff and Keil, Mark (2003) The reluctance to report bad news on troubled software projects: a theoretical model, *Information Systems Journal* 13(1), 69-96

Strati, Antonio (1999) *Organization and aesthetics*, London: SAGE

Styhre, Alexander and Sundgren, Mats (2005) *Managing Creativity in Organizations. Critique and Practices*, New York: Palgrave Macmillan

Sveningsson, Stefan and Alvesson, Mats (2003) Managing Managerial Identities: Organizational Fragmentation, Discourse and Identity Struggle, *Human Relations*, vol. 56(10), pp. 1163-1193

Osborne, Harold (1981) "What Is a Work of Art," *The British Journal of Aesthetics,* vol. 21(1) pp. 3-11

Ouchi, William G. and Maguire, Mary A. (1975) Organizational control: Two functions, *Administrative Science Quarterly,* no. 20, pp: 559-569

Ullman, Ellen (1996) Out of Time: Reflections on the Programming Life, *Educom Review* vol. 31, (4)

Weber, Steven (2004) *The Success of Open Source,* Cambridge: Harvard University Press.

Weick, Karl E. (1969/79) *The Social Psychology of Organizing,* Reading, Massachusetts: Addison-Wesley

Westenholz, Ann (2006) Identity Work and Meaning Arena: Beyond Actor/Structure and Micro/Macro Distinctions in an Empirical Analysis of IT Workers, *American Behavioral Scientist*, vol. 49(7), pp. 1015-1029

Whyte, William F. (1984) *Learning from the field: a guide from experience*, Beverly Hills, Sage Publications.

ABOUT THE AUTHOR:

Dariusz Jemielniak, Ph.D. is assistant professor of management at Kozminski Business School (Poland). He was a visiting scholar at Cornell University (2004-2005), Harvard University (2007), University of California Berkeley (2008). He recently co-edited a book on "Management Practices in High-Tech Environments" (2008), and a forthcoming "Handbook of Research on Knowledge-Intensive Organizations". His research focuses on knowledge-intensive workplace and professions, which he analyzes by the use of qualitative methods.