Through the Looking Glass of Immaterial Labor

Yunwen Ye[1] Kumiyo Nakakoji[1] Yasuhiro Yamamoto[2] Kouichi Kishida[1]
[1]Key Technology Laboratory, SRA Inc.,
2-32-8 Minami-Ikebukuro
Toshima, Tokyo 171-8513, Japan
+81-3-5979-2688
ye@sra.co.jp kumiyo@sra.co.jp
[2]Precision and Intelligence Laboratories
Tokyo Institute of Technology
4259 Nagatsuda, Midori, Yokohama, 226-8503, Japan
+81-45-924-5054
yxy@acm.org K2@sra.co.jp

ABSTRACT
Immaterial labor, which is a philosophical concept established by Maurizio Lazzarato and others for understanding the post-Fordism industry, refers to the process of producing the informational and cultural contents of a commodity. Through examining software development and software-intensive society with the lens of immaterial labor, this paper aims to make a first step of establishing a new theoretical framework to understand (1) how to evaluate values of software systems, (2) how such values are created, and (3) how software development should be organized to create such values.

Categories and Subject Descriptors
D.2.10 [**Software Engineering**]: Design – *methodologies.* H.1.1 [**Information Systems**]: Systems and Information Theory – *Value of Information.*

General Terms
Theory, Economics

Keywords
Immaterial labor, software-intensive society, valuation of software systems

1. INTRODUCTION
Makoto Hattori, the founder and director of Software Industry Association in Japan, made the following remark in 1973 [7]:

> "Most people view software development as the work of making programs, just like making products in a factory. As long as this view persists, the value of software will only be equated to the sum of labors invested in the production of the software. How we appreciate and evaluate the values, which were brought to life by strong motives, inventive resourcefulness, and thoughtful designs that software developers put into the software system, will determine the future of our industry." (originally in Japanese; translated into English by the authors)

About forty years later, the remark unfortunately remains true, and becomes even more urgent as software has permeated into every corner of our life.

The phrase *software engineering* was "deliberately chosen as being provocative," so that the phrase would stimulate conversation and dialogue [10]. After 40 years, instead of viewing the phrase stimulating and provocative, many of us have accepted a view that software is something to be engineered. We have looked up to established engineering fields to borrow concepts, theories and ideas to guide the understanding and development of the software trade. Much of research in software engineering is strongly influenced by the efficiency of Fordism, the modern industrial production that is built upon the Taylorist criteria: serialization of work, coordination of work, and insignificance of individual difference.

The engineering framework has demonstrated success as long as software to produce is something to solve a problem, for instance, in more quickly calculating a trajectory of a projectile, in more accurately simulating air dynamics, or in more efficiently searching for a phrase within a large body of text. However, more and more software systems we create today are no longer solving problems; rather, they are cultural and knowledge products that redefine the way we work, learn, communicate, and entertain. When such personal, cultural and social elements become essential in software systems we produce, the engineering framework lacks something very fundamental in software development. It is time for us to examine how software systems are produced and consumed in a very different way than other engineered commodities.

This paper uses the concept of *immaterial labor*, which is a philosophical concept established by Maurizio Lazzarato [8] and others [6, 12] for explaining the post-Fordism, to better understand the development and use of software products. The concept was first proposed by Maurizio Lazzarato who has long studied the mode of production, socialization and appreciation of culture products such as books, fine arts, audio-visual products, fashions, and other cultural activities [9].

Through examining software development and software-intensive society with the lens of immaterial labor, this paper aims to make a first step of establishing a new theoretical framework to understand (1) how to evaluate values of software systems, (2) how such values are created, and (3) how software development should be organized to create such values.

2. OVERVIEW OF IMMATERIAL LABOR
When cultural products (such as music records) become commodities, people tend to think that cultural products are another type of industrial commodities, which are resulted from subjecting intellectual labor to the norms of capitalist production.

Conference'10, Month 1–2, 2010, City, State, Country.

Lazzarato [8] argues that the other way around is happening: the mode of production, socialization and appreciation of cultural and knowledge has gradually seeped into the general economy, transforming industrial production processes into forms of immaterial labor in which "information and communication play an essential role in each stage of the process of production."

The activities performed during immaterial labor are not new, and have existed all the time. However, they were "once the privileged domain of the bourgeoisie" [8], and were not normally considered as labor because they were supported by categorically different criteria and principles [12]. What makes it important now is that more and more workers are engaged in the form of immaterial labor due to the advance of computer and software technology. Immaterial labor is not only becoming a dominant sector that replaces the industrial sector, but also becoming a predominant feature of all kinds of labor, transforming old industrial production labor into immaterial labor.

The expansion of working force that engages in immaterial labor erases the sharp dichotomy between design and making, creativity and routine work, production and consumption, and labor and leisure. As a result, it produces profound implications for the cycles of value creation, the education and qualities of workers, and the organization and management of workers.

2.1 Consumption and Production

Immaterial labor reverses the relationship between production and consumption: production creates needs, and consumption creates value. The use of the product materializes some needs that may not have existed in the first place. This recognition in turn produces more needs.

Immaterial labor does not produce for the satisfaction of known needs of consumption; instead, it creates new consumption needs. In a world of abundance, most post-modern production (such as music and fashion) is geared toward immaterial (social and cultural) wants, stimulated by producers, rather than material needs. The value created through immaterial labor tends to be in terms of emotional, interpretational, and communicational experiences.

The immaterial labor will have an economic value only when the ideas and intentions behind the product are clearly communicated to and accepted by the consumers. As Simmel points out, only until individuals are sufficiently acquainted with objects, they are able to assign their respective values [11]. Socialization is the precondition for the creation of a product value because it gives "a place in life" of the society.

2.2 Competence of Workers

The value and quality of products produced through immaterial labor depend on the knowledge and the innovation of its entire workforce. This redefines the competences of work forces in the following three aspects.

Intellectual skills and subjectivity. Cultural and information values of a product cannot be created by the mere execution of predefined procedures; instead it depends on whether workers are able to innovate by identifying problems and creating new solutions. For any given problems at each stage of production, there are often many alternative solutions, and workers are responsible to make choices based on their own criteria. This innovation and decision-making process is mainly a factor of the knowledge, taste, and personality of the worker. Intellectual skills and subjectivity, which used to be individual and private, now become the main means of production and the direct force for creating values of a product.

Communicative skills. "Immaterial labor requires cooperation and collective coordination" [8]. The quality of work is not only defined by the worker's individual professional capacities and intellectual skills, but also by his or her capabilities of initiating and managing productive cooperation with others. In addition to communicating with his or her peers, the worker also needs to be able to clearly communicate the value of the product with customers as discussed above.

Autonomy: uncertainty and motivation. Workers of immaterial labor are responsible for their own control, and to make plans and follow through. Immaterial labor cannot be divided into simple and repetitive elements. It is hardly possible for a supervisor to intervene directly how the work should be done. Autonomy requires workers have the capability of dealing with unpredictable situations and to be self-motivated in times of uncertainty.

2.3 Organizing Immaterial Labor

Taylorist principles of scientific management, which are based on the concept of planning and reducing work to simple elements to achieve efficiency, standardization and specialization, are not applicable for immaterial labor. Managers need to get out of the mentality of foreman that monitors and supervises their members. Instead, managers should work more like a facilitator, recognizing that "the autonomy and freedom of labor as the only possible form of cooperation in production [8]."

The systems theory school of organizational studies views an organization as a system consisting of inter-related and mutually dependent individual professionals who join the organization only when he or she feels the reward is fair to his or her contribution. Lazzarato calls for new approaches to organizing immaterial labor [8]. He writes: "labor and direct subjugation (to organization) no longer constitute the principal form of contractual relationship between capitalist and worker. A polymorphous self-employed autonomous work has emerged as a dominant form". Various size of productive unit could be formed for specific projects, and exist only for the duration of that job. When job is done, workers are returned to the "basin of immaterial labor."

3. IMPLICATIONS FOR SOFTWARE DEVELOPMENT AND RESEARCH

The philosophical framework of immaterial labor is relevant to software development and software industry in two respects. First, software development can be viewed as a kind of immaterial labor. Software is not made of physical material, and most of software systems we develop today redefine the way we work, learn, communicate and entertain, the values of which come into existence only after the users experience them.

The second respect, which may not be as obvious as the first one, is that software systems are the driving force that transforms material labors into immaterial labors. It is the use and consumption of software systems that characterize many labors as immaterial labors because software pushes labor activities "to the side of the production instead of being its chief actor" [12]. A large portion of employees of automobile companies and consumer electric companies are now engaged not in physical

production lines but in interacting with software systems. Software developers are not only developing tools for users, they are also changing social and productive forms for those users.

3.1 Valuation of Software Systems

The value of software, as a result of immaterial labor, is created, realized, and increased by consumers. The activity of immaterial labor that goes into the production of the software system begins to bear an economical value only when it is utilized by consumers.

Requirements are no longer something to be captured and analyzed; they are something to be innovated and designed. Software is not created to satisfy some needs that are there to be uncovered, or to model a reality computationally; in contrast, software materializes some form of vague or even non-existing needs and reshapes the reality of its users. It is not the needs that lead to production; it is the production that leads to needs.

The constantly changing requirements of users are not problems that software engineering research should aim to resolve; rather, they are the very basis for the value of software systems, and therefore represent the opportunities that should be explored and nurtured. The relative new concept of "forever beta" may be a mere reflection of the very nature of the type of software systems developed through immaterial labor.

As described in 2.1, socialization of software is the precondition for the recognition of software value. This explains the increasing new practice of software sales: releasing software with free trial times. Free trials become essential means for software developers to communicate the value of the software. Once customers recognize the value by using the product through free trials, they may be more willing to pay for their future experience. Similarly, open source software becomes the means for the software industry as whole to communicate to the society the value of software, and to generate new needs for software systems.

Software systems are not isolated products that we deliver over the fence to customers. Instead, they serve as the media to bear a social relationship between those who produce software and those who consume software. The sustained existence and success of the software industry relies very much on the new needs that continuously come from customers' usage and experience of the systems, as well as at our capability of innovation to generate and stimulate new needs.

3.2 Development of Software Systems

When viewing software as results of immaterial labor, it is easy to realize that software development is not about building computation models or representations of reality. Software is part of the reality, and software creation and consumption reshapes the reality through the creation of new modes of production while enabling new experience.

New types of software development skills and competence are called for. Lazzarato and Beller suggest that aesthetic mode of production is a starting point [9] [1]. As in all aesthetic production, the assurance of non-functional quality comes from the practice of software developers, determined by their competence and motivation of making tweaks driven by the "love of beauty and greediness for the exquisite [9]."

A few relatively new practices address such aesthetic modes of software development. The participatory design methodology and socio-technical theory have focused on realizing the quality-of-use that is only determined by users. The interaction design and experience design focus on the design of how users interact with a system while identifying necessary functionality for the system. This is in contrast with the traditional user interface design approach where an interface is inserted for the pre-determined functionality of the system.

We think these emerging design practices need to be weaved with the existing software engineering practices. A currently predominant view of software engineering research states that while software engineering focuses on building a software system correctly, what we lack is a way to build a correct system. Through the looking glass of the immaterial labor, however, there are no correct systems. The value is only created through using them. Software systems are logical artifacts. Building a software system correctly remains essential. Most of software engineering and programming practices today are still relevant to the valuation of software as a product of immaterial labor. Building software remains resource-limited and time-constrained, full of conflicting goals along multiple dimensions.

Research and education of software development is no longer limited to exhortation of the absolute good of one particular method, notation, or activity, but providing guidance and strategies that help software developer deal with uncertainties and make satisficing decisions during conflicts.

In reexamining software development from the perspective of immaterial labor, software development constitutes a design task. The design task has the duality: the design of a software system as a product (i.e., the traditional software design), and the design of the value that the software system would communicate with those who consume it. It is not that one precedes the other. It is the two faces of the one thing. The challenge of software development in the realm of immaterial labor is how to design the duality.

3.3 Organizing Software Development

In organizing software development as immaterial labor, how to sustain software developers' intrinsic motivation becomes an important research question. The quality of software systems hinges on the individual selection of alternatives, and the fusion of subjectivity and tastes. Hall et al. have identified "challenge, change, benefit, problem solving, team work, science, experiment and development practices" as motivators for software developers [5].

Hock attributed the success of the first VISA credit card clearing system to the motivation of taking pride in their work by project member: "Individuality, self-worth, ingenuity, and creativity flourished; and as they did, so did the sense of belonging to something larger than self, something beyond immediate gain and monetary gratification. [2]"

Motivation becomes essential not only for technical exploration, but also for social cooperation. While ad hoc coordination needs arise all the time, project members need to be sensitive to each other's information needs and to be motivated to help each other for collaboration to proceed timely and smoothly [13].

The notion of immaterial labor tries to differentiate two philosophic concepts of "many": *people* and *multitude*. The concept of *people* stresses the commonality and identity defined by the organization (be it state or corporation), and asks its member to converge to that organizational identity. The concept

of *multitude* takes commonality defined by general intellect as given and stresses the individuality.

Projecting these two concepts to software development, people and multitude correspond to the Roman model and the Greek model compared by Robert Glass [4]. In the Roman model, software developers identify themselves with the group, sacrificing their individuality for the good and goal of the group. In the Greek model, software developers worked as individuals and keep their individuality (Table 1). The *people* concept corresponds to the Roman model, where managers manage software developers by interchangeable roles. The *multitude* concept corresponds to the Greek model, where managers view software developers as individuals.

Table 1: Roman and Greek Models of Organization

Models	Roman	Greek
Object	Organizes people	Organizes things
Membership	Formal	Informal
Focus	Manages the projects	Writes the programs
Motivation	Motivated by group goals	Motivated by the problem at hand
Working style	Works in large organizations	Works alone or in a small group
Politic	Imperial	Democratic
Rewarding	Class based on function	Class based on merits
Communication	Planned	Ad hoc
Activities	Plan things or go to meetings	Do things

The way that open source software is developed is similar to what Lazzarrato suggested as a new form and organization for the production of immaterial labor, looking at project members as multitude. In an OSS project, each project member identifies and solves problems on her own. Coordination is bottom-up, initiated and managed by members, not by managers. Every OSS developer, to some extent, is an entrepreneur. Variations of OSS-like projects could be adopted in corporation settings [3]. Agile development methods also reflect the view of treating project members as multitude, and managing them as individuals instead of roles.

Such recent trends in organizing software development projects tend to be regarded as new styles of software development. However, by viewing software development as immaterial labor, we can see that such trends are the reflection of a better understanding of the essential features of software and the change of the nature of the software systems that are developed.

4. CONCLUDING REMARKS

If we look at the genre of business software systems, they are coming out of the first generation of productivity tools and heading into the next generation of tools for the production and reproduction of subjectivity embedded in immaterial labor in a software-intensive society. This is something similar to the change of the clothing industry that transform itself from covers that keep us warm to fashion as manifest of individuality and subjectivity. Business software of next generation is no longer for productivity alone, but for new ways of doing businesses that separate them from other competitors. Business innovations are now driven by software innovations.

Software systems are not thought products like arts; they must have utilities for their users. Software systems can be viewed as real abstraction. They are like money. Money embodies, and makes it possible to have the physical experience of, the abstract thought of “value equivalency.” Similarly, software systems embody and give experiential forms to new needs and concepts.

Immaterial labor points out the limitation of the Taylorist model, and argues that modern industry factories are increasingly becoming similar to the production of cultural products, such as fashion, music records, and software. The mode of producing software systems is becoming the mode of post-Fordism production, not the other way around. It is the time for us to stop looking up to the Ford model, stop using the factory metaphor for software development; it is the time for us to lead and define modes of production for the software-intensive society.

5. REFERENCES

[1] Beller, J., The cinematic mode of production: Attention economy and the society of the spectacle. Hanover, NH: Dartmouth College Press, 2006.

[2] Cockburn, A., "The end of software engineering and the start of economic-cooperative gaming," *Computer Science and Information Systems*, vol. 1, 2004.

[3] Dinkelacker, J., Garg, P.K., Miller, R., and Nelson, D., "Progressive open source," in *Proceedings of 24th International Conference on Software Engineering (ICSE'02)*. Orlando, FL.: ACM Press, 2002, pp. 177-186.

[4] Glass, R.L., "Greece vs. Rome: Two very different software cultures," *IEEE Software*, vol. 2006, pp. 111-112, 2006.

[5] Hall, T., Sharp, H., Beecham, S., Baddoo, N., and Robinson, H., "What do we know about developer motivation?" *IEEE Software*, vol. 2008, pp. 92-94, 2008.

[6] Hardt, M. and Negri, A., *Empire*. Cambridge, MA: Harvard University Press, 2001.

[7] Hattori, M., "SIA report," in *Japan IT chronicle*, 1973.

[8] Lazzarato, M., "Immaterial labour," in *Radical thought in italy: A potential politics*, P. Virno and M. Hardt, Eds. Minneapolis, MN: University of Minnesota Press, 1996, pp. 133-147.

[9] Lazzarato, M., "European cultural tradition and the new forms of production and circulation of knowledge," 2004.

[10] Naur, P. and Randall, B., "Software engineering: A report on a conference sponsored by the nato science committee," NATO, 1969.

[11] Simmel, G., "The metropolis and mental life," in *The sociology of georg simmel*, K. Wolff, Ed.: The Free Press, 1950.

[12] Virno, P., *A grammar of the multitude*. Cambridge, MA: The MIT Press, 2004.

[13] Ye, Y., Yamamoto, Y., and Nakakoji, K., "A socio-technical framework for supporting programmers," in *Proceedings of 2007 ACM Symposium on Foundations of Software Engineering (fse2007)*, 2007, pp. 351-360.