December 1999

Strong and Weak Programming in Information Technology

Seungkwon Jang
Sangmyung University, Republic of Korea

Follow this and additional works at: http://aisel.aisnet.org/amcis1999

Recommended Citation
http://aisel.aisnet.org/amcis1999/220

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 1999 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.
Abstract

This paper explores strong and weak thinking in computer programming and information technology. Strong thinking, which has enjoyed a high status in Western knowledge and science since the Enlightenment, emphasizes thing, outcome, and being. Weak thinking is concerned with process, action, and becoming. In information technology, strong thinking has been a dominant framework; for instance, software engineering and structured programming. By contrast, weak aspects of programming can be seen in approaches such as computer supported cooperative work (CSCW) and networking.

INTRODUCTION: WEAK AND STRONG THINKING

The last decades of the twentieth century are dominated by an increasing concern with information and communication technologies. Vattimo (1988; 1992) has summarized much of the argument about the nature of the new communication-based societies in terms of the 'strong'- 'weak' opposition. Vattimo's strong-weak opposition provides the basis for the present paper.

Vattimo's (1988) discussion of the now widely accepted distinction between modernity and postmodernity is characterized by two modes of thinking: strong thinking and weak thinking. For Vattimo, modernity is represented by strong thinking, whilst postmodernity is related to weak thinking. On the one hand, strong thinking is said to be derived from the Western metaphysical tradition that tries to capture the dynamic and the flow of experience and to replace it with static and universal criteria such as essence, stability, and truth. On the other hand, weak thinking -- in opposition to the strong thinking of Western tradition -- realizes non-categorizable processes of ambivalence, otherness, paradox, etc., which continually destroy the rigidities of strong thinking.

The same approach may be seen in the work of other thinkers such as Foucault (1977), Derrida, and Lyotard (1984), who are often cited as representatives of the postmodern (or post-structuralism) viewpoint. Post-structuralism is an intellectual movement that has led to a radical rethinking of Western-style modern civilization. Post-structuralists are concerned with re-writing the project of 'modernity' and its rationalistic conception of modern society with its emphasis on 'rationality'. Vattimo's (1988) weak thinking is an expression of the post-structuralist critique of strong thinking as an expression of modernity.

In this paper, we shall discuss the current development of information technology, particularly computer programming with regard to the weak-strong perspective: firstly, strong programming, or strong thinking, namely, software engineering and structured programming; secondly, weak programming in computer networking and computer supported cooperative work (CSCW).

STRONG PROGRAMMING

Software Engineering

The software crisis was a watershed in the history of software development; it resulted in many significant new ideas and practices. The software crisis was caused by the introduction of the new computer hardware and software systems. In the 1960s, hardware 'costs' were declining while software 'costs' were rising rapidly. Since the cost of software made up the major portion of total system costs, building huge information systems were not the same as previous practices that were simply 'scaled-up versions' of small software systems. These problems of building gigantic information systems became increasingly common, and finally introduced the debate of the 'software crisis' in the 1960s. Thus, solutions to the software crisis were needed. There have been some attempts to solve the 'software crisis', notably, software engineering. This is based on the engineering approach.

One remarkable point of software engineering is its adoption of the engineering metaphor that sees the world in terms of a 'science of construction'. In other words, programming is understood in terms of structured and organized engineering practices based on a rigorous formal approach rather than on personal skills and individual efforts. The history of software engineering can also be told in terms of 'programming styles' and the roles of programmers -- from craftsman to engineer.

Software engineering has been regarded as a solution to the 'software crisis' by which the main task of software engineers should develop and maintain the quality of software systems 'within budget' (Pressman, 1982). Therefore, software engineering should develop methods and tools to support this activity which is concerned with managing and measuring the process of the system. Software engineering represents the modernist perspective or strong thinking that foregrounds productivity, performance, and cost, etc. This explanation of software engineering is especially noteworthy because it brings out
those aspects of management and control that rest on the psychology of programmers, their productivity, group working, group leadership, and ergonomics. These approaches aim to achieve the good management of programming in order to get over the software crisis.

Like any other engineering discipline, software engineering can be seen as a typical 'rationalistic' approach to the problems of computer programming. It is significant that the inventors of software engineering employ the terms 'engineering' and suggest that information in the modern world can be constructed and produced like industrial products. To sum up these accounts of the software crisis and software engineering, we can conclude that software engineering is the 'rational' solution to the software crisis the ways in which they employ 'engineering metaphor'.

Software engineering can be viewed as a modernist perspective in computer science. Engineering solutions look mainly for improvements in performativity and productivity. Hence, the engineering approaches in software development are said to be strong programming.

**Structured Programming**

Like software engineering, structured programming as a programming methodology is also deeply implicated with strong information inasmuch as writing 'structured programs' means to construct strong information and in particular to engineer strong information from weak information.

To overcome the software crisis, programs should be 'good' or 'better' with respect to maintainability and controllability so as to reduce cost, while at the same time keeping programs working. Good programs are easy to read, easy to understand, and easy to modify so that structures of programs become the key to the management of large software systems. Thus, logically and functionally structured and ordered programs are essential. In other words, the whole point of a good program is to get rid of weak and fuzzy structures. For this reason, the skill of programming is said to be nothing but the art of organizing 'complexity' and 'chaos' (Sethi, 1989).

As far as programming methodologies are concerned, structured programming is the most influential development since the 1960s. Keuffel (1991) suggests the reason why structured programming has had such a powerful impact in the programming world: structured programming has been developed along with software engineering. Generally speaking, before the emergence of software engineering, programming was just regarded as an undisciplined art or craftsmanship. But, software engineering made programming the serious engineering works. Thus, the movement of software engineering appears to be a driving force of structured programming method. Based on the history of structured programming, we may connect software engineering with programming activities more closely.

Structured programming is closely associated with software engineering in many ways in which they all are concerned with cost and reliability. In this way, the legitimacy of structured programming lies in achieving the greater productivity that has been the primary goal of software engineering. Thus, both structured programming and software engineering can be categorized as strong programming in terms of their ideological orientation and the foundation of legitimacy (i.e., rationalization for efficiency). Strong programming that is 'writing' a program with the attributions such as controllable, readable, and reliable has been the aim of programming methodology.

**WEAK PROGRAMMING**

**Networks and Weak Programming**

What, then, is weak programming? To answer this question, we need a critical re-thinking of programming. The etymological meaning of programming is 'writing-beforehand' which should be understood as 'strong programming' in the sense that it means to prefix later happenings, and therefore the ability to predict and control. However, viewing programming as writing creates an entirely new perspective on programming; it shows that strong programming has a 'weak' shadow or underside, which is normally ignored.

In the context of weak thinking, networking produces the conditions for weak programming. It might be true to say that activities of computer networking are initially 'strong' because it aims to be strong programming with regard to its functions, i.e., developing effective environments of programming and so on. But as soon as strong programming of networking establishes the 'right' place (networks), strong programming is going to be 'weak' because the characteristics of strong programming themselves become to destroy their foundation. Strong programming as writing and inscription is weakened by the very nature of writing as difference and separation.

Notably, recent developments in computer networking illustrate this phenomenon very well. Initially, computer networks are designed in order to attain a desired and predictable status. However, soon after networking emerges, it is no longer a controllable artifact which is planned by the heads of hierarchies. The computer network is interconnected or 'intertextualized' very closely with other computer networks, and moreover it is also interwoven with social networks. In this regard, programming becomes an interconnecting and 'intertextualizing' process in the network. This can underpin the idea of weak programming as 'dissemination'. Computer programming makes it possible to disperse and distribute programmers and users throughout computer networks.

Networking could be a precondition of weak programming in the sense that networks provide new ways of working practices such as working in distance and computer-mediated communication. Hence, the idea of working in identifiable time/space is no long valid even in many conventional working practices as well as computer programming.
Computer Supported Cooperative Work (CSCW)

CSCW is a new discipline that is concerned with cooperative working under the computer-based working environment, whilst groupware as software is another facet of CSCW. Groupware or CSCW is a resourceful example of weak programming in the 1990s.

It is argued that among the paradigms of computing, networking is a main paradigm of the 1990s (Tesler, 1991). In this paradigm, groups (not individuals) are the main users; moreover, the objective of computing is not to 'present' but to 'communicate'. Hence, networking becomes an important technical condition or requirement of CSCW that is very significant development of this decade.

It has been said that a technical marriage between computers and other forms of communication forms the basis of CSCW (Ellis, Gibbs, and Rein, 1991). CSCW researchers focus on the ways in which group can work and on how technology can help group work. In this context, CSCW research recognizes the significance of 'communication, collaboration, and coordination'.

The aim of CSCW is to provide integrated support to 'user groups' in three main facets of work: face-to-face activity, activity at different times, and activity in different geographical locations (Wilson, 1991). In CSCW, the time-space dimension is very important in building systems. In addition, CSCW researchers are concerned with multiple media of communication: for instance, the use of voice, image and video; multi-party desktop video conferencing; intelligent agents; network resources.

CSCW typically symbolizes computer developments in the contemporary world to such an extent that we need a new worldview to substitute weak thinking for strong thinking. Thus, we can now take the meanings of information technology more seriously in terms of weak thinking such as weak programming.

CONCLUSION AND FURTHER RESEARCH DIRECTIONS

Strong thinking is still dominant in software development such as software engineering and structured programming. However, after a few decades of development, software system development is no longer totally subjugated by strong thinking. Rather, weak thinking in software development such as CSCW is becoming a widespread trend.

It seems to be we have discussed simply two contradictory ways of programming in software development. Nevertheless, there are important issues not to be discussed yet in this paper. These issues include:

- Should weak and strong programming be viewed as a simple binary division? If not, what is the nature of relationship between weak and strong programming?
- Is the strong programming style (such as software engineering) a product of the competition in computer discourses?
- Can we take the point that discourse produces knowledge out of struggle rather than by harmonious consensus? And, can knowledge construction be seen as an outcome of 'agonistics', that is, the struggle between actors?

REFERENCES


