Scandinavian Journal of Information Systems

Volume 3 | Issue 1

Article 2

1991

PHILOSOPHY AND INDUSTRIALIZED SOFTWARE DEVELOPMENT

Ole Hanseth

Norwegian Computing Center, Ole.Hanseth@nr.no

Scandinavian Journal of Information Systems, Vol. 3, pp. 111–130, 1991

PHILOSOPHY AND INDUSTRIALIZED SOFTWARE DEVELOPMENT

OLE HANSETH
Norwegian Computing Center
P.O.Box 114 Blindern, N-0314 Oslo, Norway

Abstract

In this paper I will give a contribution to the creative conversation on future development and use of computer technology which Bo Dahlbom and Lars-Erik Janlert invited to in their paper published in the previous volume of SJIS.They argued that as software development now becomes industrialized, the need for organizational competence in this task will disappear. My position is rather the direct opposite: Software development will to a larger extent be taken over by industry. This implies, however, that the software systems being developed have to satisfy the requirements of several organizations, not only one as is the case in in-house development. And further, the software has to be integrated with several existing systems and components. Accordingly, future software development will demand more competent and more interdisciplinary development teams.

I will argue that there will, in the future, be a huge demand for computer scientists with a thorough knowledge about organizations. Knowledge from both philosophy and the social sciences will be important. Industrialization of software development, more strategic use of software, and more integrated information systems are important trends causing this demand. Accordingly, I find the systems development educations in Scandinavia, criticized by Dahlbom and Janlert, highly relevant.

1 Introduction

In this paper I will give a contribution to the creative conversation on future development and use of computer technology which Bo Dahlbom and Lars-Erik Janlert invited to in their paper published in the previous volume of Scandinavian Journal of Information Systems (Dahlbom 1990b). I will try to do that by making a critical note on the view on software, software use, software development, and the Scandinavian research tradition in systems development given in this above mentioned paper and those presented by Bo Dahlbom at the workshop on "Software Development and Reality Construction" in West Germany in October 1988 (Dahlbom 1988), and at the Sydpol seminar on "Organizational Competence in Systems Development" (Dahlbom 1990a). The critical note will be based on my personal experience from practical software development tasks, in particular three projects, as well as my more theoretical focused research in computer science.

Dahlbom's view is: As software development now becomes industrialized, the need for organizational competence in this task will disappear. My position is rather the direct opposite: Software development will to a larger extent be taken over by industry. This implies, however, that the software systems being developed have to satisfy the requirements of several organizations, not only one as is the case in in-house development. And further, the software has to be integrated with several existing systems and components. Accordingly, future software development will demand more competent and more interdisciplinary development teams.

2 Dahlbom's view[1]

This section presents those of Dahlbom's claims that I will discuss, and the outline of the paper.

Dahlbom claims that systems development practices:

> is currently undergoing a process of industrialization. From having had the character of consulting in short term projects, ..., system work is changing into being (1) a permanent part of maintenance, (2) a part of the software industry, (3) done by end-users, (4) product information, and (5) general, non-technical organizational development.
>
> Scandinavian research on system development has been much too slow in its reaction to these changes. The discussion of system development practice is still dominated by a conception of it as 'project work,' and major efforts are spent on 'project design.' But there will, in a few years from now, be no place for detailed system analysis and design work in the field. *The typical 'system designer' will be assembling ready made products.* (my italics). the engineering consultants are entering the software laboratories and the organization-oriented

> consultants become advanced sales persons or general consultants ... moving in the direction of (2) and (5) above. (Dahlbom 1990a, p. 143; 1990b, p. 96–97)

Dahlbom assumes further that the industrialization of software development is leading to the adoption of traditional industrial production processes and the same deskilling of its employees as other industrialization processes.

This is, according to my view, both right and wrong. Current and future software development will essentially either be based on existing, standardized software components, or integration of existing systems, or both. But software development based on standardized, existing components is much more than just putting readymade pieces together. I will document this, in section three of the paper, with our experience from the development of an ODA-based distributed document archive based on standard components. When putting several such pieces together the total system becomes very large and demands a technical competence beyond those needed for developing traditional 'stand alone' systems. I will also argue that industrial systems development will be more complex in all respects than traditional in-house development projects, demanding more competent and more interdisciplinary development teams.

As software development is industrialized the important questions are, according to Dahlbom, related to use of computer systems:

> But all that is changing now that more and more of us begin to realize that the exciting future of information technology research lies in questions of use ... Future system developers will need a rich picture of the possibilities and varieties of human-computer interaction ... There is a danger that industrialization will shape a corps of system developers who see their job as working with a plethora of hardware and software on one side and with user organizations on the other, and it will be her job to make the twain meet. She'll be an expert on what to do with all the artifacts produced by computer industries, rather than on how to produce those artifacts. (Dahlbom 1990b, p. 97)

I agree that use of computer systems is a topic of increasing importance. But looking at use of computer technology we will also see trends running counter to this industrialization process. I will, in section four, look at one way of using this technology, namely as means for obtaining a strategic advantage in relation to competitors. This can, for instance, be done by putting a piece of software into the product, making it unique. To do that one also needs a unique system, not exactly the same as all competitors. Such development tasks demand highly qualified technical people working closely together with the managers responsible for the organizations' strategies.

Dahlbom draws the conclusion that one should base:

> education in system development on a research approach to organizations founded on a detailed study of the nature of information techno-logy use. (Dahlbom 1990a, p. 143)

Knowledge of philosophy and social sciences, which systems development researchers in Scandinavia have been concerned about, is completely irrelevant:

> to go on about tacit knowledge in the 90's will be as relevant to the practice of systems development as Hegel's comments were to the practicing electrician.[2] (Dahlbom 1990b, p. 86)

I agree that knowledge about information technology use will be important, but I will argue in section five of the paper that what is needed for industrialized software development and strategic use of integrated software systems, is people with high technical competence and thorough knowledge on a wide range of social and philosophical topics working together in multi-disciplinary teams.

One 'use need' is that of integrating existing systems, for instance through the establishment of shared data models/data bases, or by means of EDI. Such integration tasks will often be very complex, both from a technical and an organizational point of view. When designing huge shared data models one risks running into the same kind of trouble as when trying to create AI and natural language understanding systems. (This is documented by Ronald Stamper (1984).)

Dahlbom gives no empirical evidence in support of his claims regarding the nature and consequences of the industrialization of software development. This fact, in itself, makes his claims no easier to believe.

3 The Industrialization of Software Development

In this section, I will analyze industrial software development and its implications concerning needs for competence.

Software people have been talking about the software crises, referring to the costs and common failures of software development projects, for nearly twenty years. The problems have often been considered to be how software development projects are organized. Industrial methods in form of a Tayloristic assembly line in combination with methods (like object oriented programming) for reusing existing components, have regularly been proposed as a panacea making the problem evaporate (Cox 1990). These methods will lead the industrial revolution to the software field. This is, to my view, simply wrong.

3.1 Industrialization and maturing of Technology

As long as software has been developed, parts of it has been developed as products by the software industry. Accordingly, in some sense software development has always been partly industrialized. Early software products were operating systems, assemblers and compilers. Later on the basic systems like data bases and communication protocols appeared. Also the development and maintenance of end user products such as text processors, spreadsheets, systems for numerical and statistical analysis, accounting systems, and production control systems have basically been taken over by software industry. Accordingly, Dahlbom is correct as the development and maintenance of some categories of software systems has been partly taken over by the software industry. Concurrently, however, new technology and applications have popped up. There is no reason why innovation should stop in this field.

How a software system is usually developed depends on the maturity of the technology it is based on. J Birnbaum splits development of what he calls pervasive technology into four phases (Birnbaum 1985). The characteristics for each of them are:

1. experimental rarity, laboratory curiosity,
2. exotic tool or toy,
3. produced in a number, and
4. integrated into daily life, (as, for instance, television today).

Birnbaum says that in the third phase the products are proliferating. There are a great number of them although many are looking similar. Characteristic for phase four is a consolidation within the businesses; innovations are much harder because of large investments, and the prices are stabilizing. As an example Birnbaum mentions the automobile industry. During its first phase there were about 1100 car companies in USA, today this is reduced to 3 1/2 and the prices and models are very similar.

I think that Birnbaum's theory can account pretty well for the development of software technology. However, as the existing technology has matured, new immature ones have appeared and been applied to new areas. Examples of currently rather immature software technology and applications are hypermedia and multi-media technology; object oriented data base management systems; expert systems (in particular tools for developing such systems); highly parallel architectures; "groupware" and other kinds of integrated distributed systems. Applications in some areas as e.g. patient care (hospital medical record systems) are immature too.

3.2 Industrialized Production and Pmmature Technology

I think Dahlbom's claims may also be correct in the sense that the development of systems based on rather immature technology has partly been taken over by the software industry. He does not give any description of how he thinks the software development in the software industry is carried out. But he says that the people working there will go through the same deskilling processes as other types of craft work when being industrialized. There will only be a need for limited technological knowledge. Reading his papers I get the impression that the work will be carried out as traditional assembly work, "assembling ready made components." And he also seems to believe that all software will be developed and maintained by the equally structured industrial production processes.

Dahlbom talks about software development as if there is only one single piece of software technology, or that all kinds of software technology is equally mature, and that all kinds of software systems are/will be developed by exactly the same industrial production processes. Referring to the section above, there will always be new immature technology.

It is roughly correct that software development in general is, apparently, undergoing a process of industrialization. It is correct in the sense that a larger part of software development is carried out by software companies rather than by in-house projects. It is also partially correct in the sense that traditional industrial (Tayloristic) production processes are adopted as far as maintenance and development of new versions of existing products based on mature technology are concerned, and when the products are applied within well computerized application fields. But it is definitely not correct in the latter sense concerning the development of applications based on rather immature technology and for less computerized fields.

Our knowledge about how to apply immature technology is limited. Due to this there cannot be any prescribed procedure, like an assembly line, for how new systems based on such technology should be developed. The development process has to be exploratory with focus on learning how to apply, and maybe also improve the technology to fit the task at hand. The essence of the maturing process is to develop the knowledge necessary to effectively produce systems applying the technology.

Development of software systems will *always* be the development of something new, different from anything existing, otherwise it would not be developed. (I will return to the similarities and differences between software development and industrial mass production and engineering in Section 5.1). It will always be a design process in which we have to learn what is a good design and what is not. Traditional mass production is aimed at making a large number of exact copies of a prototype. When we want an exact copy of a software system, we produce it by just executing the copy command on a computer where the system exists, copying all its files. In this way the software production process, which may be directly related to traditional industrial mass production, is already fully automated. Development of new generations of software systems will be closer to the development of new car models than the production of the different instances of this model. And while car production is becoming more and more structured and automated, the process of developing new car models is growing, both in costs and extent.

As an argument for the development towards highly industrial production of software systems, Dahlbom compares clock-making today and in the 14th century.

Although he is wrong in the description of the short term changes of software development processes, he may be right if he is describing software development in 600 years. But I do not find that time horizon of particular relevance.

3.3 Software Development by Reusing Standardized Components

An important aspect of industrial production is usually considered to be the production of new products by putting together readymade components. Related to software development, this phenomenon is usually termed reuse of software. It has been considered to be an important topic since, at least, the early sixties, when the development of languages and tools to support the use of procedure/program libraries started. And existing software components have been reused in a continually increasing degree since that time. However, reuse has taken on an extra dimension in the later years through standardization efforts and the 'convergence in the market' for several product categories[3] as a result of the maturity level the technology has reached. If by industrial production Dahlbom means reuse of existing components, it is true that software development has become industrialized. But if he thinks that using standardized components makes the software development simple and that it can be organized like assembly line production, he is certainly wrong. I will argue for this claim by describing two development activities in which I have personally been involved. Although they are closely related, the first will focus on software development based on standardized existing components, the latter on software product development in an industrial setting.

In a project we developed a prototype of a document archiving system (Larsen 89). Documents were assembled in a hierarchical folder (file) system. Both were stored in a data base - the documents in ODA[4] format. The documents could be exchanged between users by means of an electronic mail system, and converted between ODA and the private format of the (multi media) document editor the user preferred for browsing or editing a document. The prototype system was implemented by using standard readymade components as far as possible. We used a mail system implementing the X.400[5] standard; a relational data base providing a standard SQL[6] interface; standard operating system - UNIX; standard programming language - C; standard format for document exchange - ODA; standard windowing system - SunView; standard tool for data encoding/decoding - ISODE; and an ODA - Diamond converter developed by University College London.

When putting together components like those mentioned above, the potential for fast development of powerful systems is large. But to make a new system, one has to design and implement a piece of software integrating all the components into a functioning totality. This totality becomes very complex making the integration work really hard. To design and implement the system up to the point where we tested all relevant combinations of two and two components together was a rather straightforward task. But to integrate the whole system was quite another one. The most complex problem was caused by the huge number of identifier name collisions when linking all the modules together. Identical symbols were defined in different modules. This problem is a bit tricky as it is closely linked to a C/UNIX linker feature. One will often redefine system-implemented functions, and the private version will be used if the programmer makes the linker find this one first. Accordingly, there will, in most cases, be lots of intended name-collisions when linking C programs. When linking our system in the original version, approximately 61.000 different symbols were visible to the linker. To find those that were defined several times would not be quite simple. To find those that were correctly redefined and those that were not in this group again would be impossible.

What we did to make the prototype running was to put on an extra level of encapsulation, hiding more symbols and splitting the system into several UNIX processes being linked separately, and thereby slowing down the performance of the system.[7]

To me, the lesson to be learned from this case is that by using readymade components one can develop larger systems with less effort, but the systems become technically more complex. Accordingly, developing software systems this way demands software people with *higher* technical competence than earlier. Assembling ready made, standardized components is not as easy as Dahlbom seems to believe!

3.4 Industrial Development of Systems for Health Care

The next example I will use to illustrate current industrialized software development processes is the development of a communication system for exchange of data between health care information systems.[8]

The document handling system mentioned above was later on used as the basis for a prototype system for exchange of medical information.[9] This prototype is one of the important sources of knowledge for the development of the communication system as an industrial product. It will be developed according to a concept of 'independent but cooperating systems' distributed on a number of computers. One system will cooperate with others according to a client/server model realized by the communication system. They will communicate with each other; with existing mainframe based InfoMedica systems; with systems from other vendors; with systems in the primary health sector; and with systems outside the health sector such as social security.

The system will follow international standards for exchange of medical information (Medix, HL-7); it will be based on standard communication protocols (OSI and Internet); run under primarily, but not exclusively, 'open systems' (UNIX, MS-DOS, OS-2). This system will be developed by a member of the software industry, but the development process will certainly not be like a typical industrial assembly line production process. And I think this example is a fairly representative one for the way the software industry will develop new software products for banking; the health sector; the public sector; insurance; and many other application fields for quite a long time. There will be exceptions of course, technical systems like data bases and compilers, and general mature systems like word processors and spreadsheets.

These kinds of industrial software development processes will be more complicated than traditional in-house software development in several respects. We are going to develop a general product. However, as we are going to develop a new system we have to go through the same prototyping and learning cycles as in in-house development projects. And - before a system will work at all sites it has to work at one. Accordingly the first version of the product must be developed together with at least one pilot customer as if it was close to a traditional project. In the development of this pilot version of the product we will face all the problems of traditional systems development projects — *and many more!*

These additional problems can be described as additional constraints on the development process. At the same time as the system shall satisfy the needs of the pilot organization it has to be easy to adapt to the needs of others. The system development process will be more complicated due to the different and often conflicting goals of the software company and the pilot organization. The technical aspects of the process will also be more complicated as the product shall be easily scaled to a product existing in several variants which can run on different operating systems; computers; communication protocols; together with different kinds of applications; different windowing systems; different data bases; etc.

The process is also made more complicated as several products will be developed concurrently in cooperation with different pilot organizations. The development work needs to be tightly coordinated as the systems must be easy to integrate; one would like to make use of the experiences from the other projects, one should use the same tools; the user interfaces should be as equal as possible; etc.

Buying a product from a software company does not imply that the product is ready for use. In 1986 a Norwegian insurance company changed their product portfolio, demanding one of their system to be changed (Hanseth 1988). They found that this system was impossible to modify as needed. A new one had to be developed. This started a chain reaction. The new system demanded changes in another beyond its modifiability. This process resulted in the unavoidable decision stating that all their systems had to be replaced. In-house development was estimated to about 100 mill. NOK. However, they found a product delivered by an American company which they considered to be acceptable. This product was bought for 6 mill NOK. The tailoring of the system to the company's products and production processes costed approximately 45 mill. NOK!

Tailorability has been a popular term for some years, and Dahlbom says that in the future "Hardware and software industry will produce ready-made systems solutions, and there will be standardization both on the product end and on the user end. To the extent that the technology will encourage individual tailoring, the competence for such tailoring will be widespread skill" (Dahlbom 1990a, p. 144). This may be true for simpler systems (Carter 90) but not for the kind of complex systems we are talking about here. To the extent this claim will ever be true for such systems, its time horizon is beyond any serious interest. This claim will be supported by the arguments of the next section.

3.5 Conclusion

I agree with Dahlbom that software development is becoming more industrialized, both the development of technological mature systems, and the development of systems based on less mature technology and applied within non-computerized fields. To me, this fact has two important implications:

1. The total software development costs can decrease as they are 'shared' among several user organizations.
2. The software development process becomes *more complex* as the systems have to satisfy the needs of a large number of organizations.

The latter conclusion implies that industrialized software development presupposes multi-disciplinary teams of people with higher competence than those doing the development work today. This argument will lose its power when there are no more fields to computerize. Currently, I cannot see that we are close to reaching that point. And organizations and work processes will change and thereby generate needs for new systems.

4 Strategic Use of Software Systems

Dahlbom argues for the importance of studying use of computer systems. I agree that this is an important and much neglected problem. However, there is one very important and much discussed aspect which I do not find in his treatise on the topic: the strategic role played by a system or *strategic use of information technology.*

The usual application of computer systems is to speed up and thereby lower the costs of some production process. Strategic use of technology denotes the conscious linking of technological opportunities with overall business strategies. Joseph Morone found that innovative firms were distinguished by their ability to translate awareness of technological development into strategic options (Morone 1989). Computer technology is currently a very important technology in this respect (Benjamin 1988). Strategic use of information technology is to utilize this technology in a *unique* way; to develop a unique product or a unique production process. Such uniqueness will in most cases demand unique systems! Maybe it is possible to obtain a strategic advantage by just buying an existing software product and put some unique data into it, or utilize an existing system in a unique way. But in general it will be hard to obtain this by means of a system which is used by most of the competitors.

From this line of reasoning we can conclude that there will still be a demand for development of tailormade systems. And this development will demand people with competence across a wide range of technical as well as social and humanistic disciplines. Joseph Morone claims that firms making strategic use of technology exhibit strong internal technical capabilities which are (a) loosely coupled to current operations, and (b) tightly coupled to strategic decision making (Morone 1989, p. 103). He also holds that firms that succeed in using technology for strategic advantage exhibit a high degree of learning from experience. Successful new products and processes emerge gradually, over the source of a sequence of earlier product and process introductions (Morone 1989, p. 105). These are also strong arguments for current and future needs for teams with high multi-disciplinary competence working closely together. In these teams software people with high technical competence combined with an understanding of organizatio nal and political processes will play key roles.

Benjamin and Scott Morton say that *interconnection* and *integration* are the important keywords describing how to use computer technology strategically (Benjamin 1988). Interconnection can be improved with respect to: (1) cost of transmission; (2) purity of the data; (3) speed of access; (4) type of data (for example numeric, text, and graphics); and (5) availability of data (large public data bases and electronic message systems). Integration has been enabled by improvements in interconnectivity and data accessibility. The combined effect of several forms of integration is to fundamentally alter the way the organization does its basic work. Some specific forms of integration are: (1) the integration of multiple-transaction classes to present a common interface to the user, (2) the integration of multiple forms of representation into a single form; (3) the integration of expert knowledge to provide a standardized process for accomplishing or supporting tasks; and (4) the integration of groups into new forms of problem-solving networks, through electronic messaging and conferencing, voice messaging, and video conferencing implementations (Benjamin 1988, p. 93–94). These integration mechanisms may, combined with a restructuring of the organization, lead to a repositioning in the market.

Benjamin and Scott Morton give some examples illustrating this integration, restructuring, and repositioning. The examples cover financial investment management; airline reservation; engineering and manufacturing financial services; and product configuration.

Just as software development is becoming more industrialized, strategic use of software systems is increasing. Because unique systems are required, it will still be necessary for organizations to develop parts of their systems, in particular the integrating parts, themselves. This development towards more strategic use of software systems has two important consequences:

1. A significant part of an organization's systems will be developed in-house.

2. The development of strategic, integrated systems will in general be more complicated than today's development processes.

Accordingly, strategic use of software technology demands more competent multidisciplinary teams working closely together. And design and use will be closely related to each other.

On the basis of my analysis in this section and the previous one, I conclude that Dahlbom's claims about the need for organization competence in software development is completely wrong. He argues that knowledge from the social sciences and philosophy has become irrelevant due to the industrialization of software development. Because of the complexity of industrial software development and the growing importance of strategic use of software, I find the importance of such knowledge to be growing.

5 The Need for Philosophical Knowledge in Industrialized Software Development

I will here, after some general comments, discuss two areas illustrating the need for philosophical knowledge: software development research and integration of data and information systems.

5.1 Introduction

The philosopher Dahlbom advises computer scientists to forget philosophy and social science, and concentrate on what, according to his view, the important problems within the computer science (systems development) field is: The development of technologically based views of organizations especially designed for understanding use of computer technology (and technological work in the industry). An important topic for a conversation between a philosopher and systems development people could be philosophical problems in software development. I will here illustrate some problems and argue for their importance. Although I find social science equally important, I will not those matters.

Dahlbom finds that the "new paradigm" approaches based on philosophy and social science, are either wrong or impossible to apply. I agree that the solution is not to let the 'new paradigm' replace the old one. What I think we need, however, is a 'framewor' or 'paradigm' which rather embed the technical approach, and which can help us to understand how we should use it. (I will return to this point at the end of this section.) I find the efforts to work out an approach based on philosophy and social sciences (Mathiassen 1981, Ehn 1987), which Dahlbom criticize, very important steps in this direction.

The interest for philosophical questions among software and systems development is, I think, to a very large extent a product of Hubert Dreyfus' criticism of the basic assumptions of artificial intelligence, a criticism based on the philosophy of Heidegger, Wittgenstein, and Merleau-Ponty (Dreyfus 1972, 1986).

This work was followed up by Winograd and Flores who broadened the scope of philosophy, in particular hermeneutics and speech act theory, to design of software systems in general (Winograd 1986a). Pelle Ehn (1987) continued by linking philosophy of Wittgenstein and Heidegger to the social sciences and apply it to systems development work. To me, the work of these people and others has provide very important knowledge about what software development really is.

Concepts of central importance in computer science have been analyzed and discussed in detail by philosophers, and one would do well to look into that discussion before rushing into building systems. Knowledge is one such concept, language is another one.

Concerning all the failed efforts to build useful knowledge based systems, it appears to be in the best scientific traditions to look into the theory of knowledge to understand the nature of these problems, failures, or refutations. One should question the 'knowledge as a thing' approach. In this approach, knowledge is something which can be manipulated independent of any context by a person not understanding it and without affecting its meaning. Knowledge can be extracted from an expert, transformed from one representation into another, put into a machine, etc. A philosopher could help the engineers understanding of the presupposition and limits of this view, as well as other alternatives like Wittgenstein who held that knowledge cannot be independent of the situations in which it is *used.*

Terry Winograd has argued extensively for the relevance of speech act theory for systems development (Winograd 1986a, 1986b, 1987). In the next section I will try to illustrate how the philosophy of language, in the form of a 'Wittgensteinian semantics,' may help us to understand integration problems.

5.2 Software Development Research

Dahlbom asks why some people find it so important to stress that systems development work is not just an ordinary engineering activity, and accordingly needs other kinds of knowledge. He proposes four answers,[10] and rejects all. I will here give some others which also illustrate the need for philosophical knowledge.

Software development is different from traditional industrial production and engineering disciplines because of the nature of software considered as a 'material.' It can be changed at any time during its development and life time. As mentioned in Section 3.2, it is very easy to make a new example of a product type by just copying a number of files. Tayloristic production processes in industry is designed for making a large number of exact copies of a product. The stucturedness of a problem can be conceived as determined by three factors: its complexity, its degree of novelty, and its ambiguity (i.e. if one can determine whether a proposed solution really is a solution or not) (Kaufmann 1988). Mass production is characterized by relatively high complexity. But there is absolutely no degree of novelty as the task is to repeat already executed processes, and there is no ambiguity.

Engineering tasks are also characterized by high complexity, a low degree of novelty, and a low degree of ambiguity. The building of one particular kind of platform for oil drilling at sea, for instance, is almost equally difficult the second and third time as the first.

Software development is characterized by a high degree of novelty as well as ambiguity. The degree of novelty varies for different development projects of course (due to the varying degree of maturity of the technology applied), but it is regularly high as we always develop unique systems. If we just want a copy of an existing one, we buy one. The degree of ambiguity is also regularly high as the users do not know what kind of system they should have and the target organization as well as other environmental factors change during the development (Curtis 1988).

More strategic use of software will contribute to keep the degree of novelty and ambiguity of software development tasks high.

As software development is different from engineering and production work processes, we cannot copy these processes blindly. The high degree of novelty implies that there is no given procedure to follow. The process must be exploratory, paying attention to learning and the changing conditions. We have to search outside engineering and production principles to find effective processes for software and systems development. And this search is an important task for systems development research.

An important part of science is to construct theories *explaining* empirical observed phenomena (Elster 1983). Of special interests are the phenomena which cannot be properly accounted for by existing theories. Research in systems development should then be concerned with building theories of systems development work and *improving* these by searching for explanations of the phenomena left unexplained by the current theories. I will here argue for the claim that a major weakness of research in computer science, software engineering, and systems development is its *unscientific* tradition. Rather than doing research according to scientific methods one is concerned about inventing 'smart things' (i.e. systems or methods) which can originate a huge business. Even the most advanced research in the field is, with a few exceptions, closely linked to business or military interests.

Popper's concepts 'conjectures and refutations' from the theory of knowledge seem useful for computer scientists and system developers (Popper 1989). According to Popper a theory can never be verified. Irrespective of how many observations we have which conform to the theory, we cannot know that there will be no counterexample. Accordingly a theory cannot be verified, only falsified when we find a counterexample. Our (theoretical) knowledge grows when we can construct a new theory which explains this, and not by finding yet another example conforming to the theory.

In software development, research as well as practical work, failures are usually considered as something which do not appear if one does proper work. The approach followed is taken for granted to be close to perfect. Failures are explained as either due to have chosen the wrong method, not following the method properly, or accidental external (social/political) circumstances. Absence of failures are explained by following the correct method properly. This phenomenon is found to be shared by most disciplines of technology and natural science (Pfaffenberger 1988, Gilbert 1984). The shortcomings of an approach is not admitted before a new one is established and all its necessary marketing propaganda and rhetorics is "worked out". An important role of this rhetorics is to hide the problems of the failed approach so that nobody, hopefully, reminds the practitioners within the field that they had 'failed.' In this way, what should serve as the sources of new important knowledge is 'destroyed.' An example is the replacement of the 'general problem solver' approach by the 'knowledge based' one in AI in the later sixties. When the first approach was given up, the explanation of its failure (refutation) was worked out only as far as it served as powerful rhetorics and propaganda for the new one. "A general problem solver is impossible to build, one needs domain specific knowledge." All efforts were directed towards building knowledge based intelligent things — none towards getting a more thorough understanding of why the old approach failed.

There are of course, many other examples both from the AI field and all the others. I find the lack of concern about scientific method and theory of science in software development research to be a serious fault. The Scandinavian systems development research is one of the few examples I know where one focuses seriously on the problems encountered and tries to work out a thorough understanding of these. And to search for knowledge from philosophy and the social sciences to help explain the problems is just serious scientific work. And I believe a philosopher can help to improve the practice of software development research as well as other areas of computer science research.

I am not arguing for a replacement of a practical, engineering oriented research tradition seeking practical solutions that work by a pure scientific tradition seeking only truth. The concern about relevance inherent in the practical oriented tradition is very important to preserve. But in my opinion this practice can be improved significantly by paying attention to some epistemological questions.

5.3 Integration[11]

In previous sections I have stressed the importance of system integration. I believe we have lots of hard integration work awaiting. Integration is usually a technical condition for strategic use of software systems. Integrating systems across different organizations by means of EDI (Electronic Data Interchange) is currently a very hot topic. Large organizations will try to integrate their systems through communication systems like the healthcare information system mentioned in Section 3.4.

Integration is as old as software development itself - and with lots of failures. These failures are hardly given any explanation at all, except that integration is very hard, and that integrated systems usually become too complex. An important example of such an integration effort is the MIS — Management Information System — concept from the seventies. When the companies installed both accounting and production control systems, it was discovered that they partly used the same data. This data should be registered only once and stored in only one place. All the systems should be integrated around one shared corporate data base. Although some shared data bases exist, the MIS theory turned out to be a failure. Why? Because the data bases became to complex! This explanation is a bit incomplete. If we intend to succeed in our new integration efforts we will certainly need a better one.

Ronald Stamper has described one integration effort failure in some detail (Stamper 1984). The system integration project in fact succeeded as far as the technical implementation is concerned. But it failed in *use*. In a British municipality the data base of the information system of the social workers was integrated with that of the urban planners. Using the system however, they discovered that "unfit for habitation," which was a possible value of an attribute, meant one thing in one institutional setting and something quite different in the other.

The Norwegian psycholinguist Ragnar Rommetveit has provided an explanation of this failure/example based on Wittgenstein's theory of language saying that meaning is defined by use (Rommetveit 1983, 1986). A term is used to discriminate among the phenomena included in a context. The same term denotes different phenomena in different contexts, and the same phenomenon may be denoted by different terms in different contexts. Rommetveit applies Wittgenstein's theory to information systems enabling him to explain this integration problem as caused by the developers ignorance for the context dependence of the data. Wittgenstein's concept of language games in which the rules are made 'as we go along,' can also explain the evolving meaning of data beyond their context dependence.

In our preliminary effort to design a communication system enabling an integration of applications in the health sector, we have found some examples of this context dependency of data. A large number of physicians in the primary health care use a computer-based patient record system. The systems are also used to support the production of the physicians' documents. It is proposed that the system should extract the relevant data from the patient record and transmit them to those who need them. Some data should be extracted to constitute the bills. Some others to provide the social security system with the data necessary to determine whether the patient herself should pay the medicine bought from the pharmacy, others should be sent to administrative offices to generate statistics.

One key data element in these documents/data sets is the term denoting the patient's diagnosis. The diagnoses is registered in the patients' records. And the diagnosis is a parameter determining whether the patient should pay the medicine herself or not as well as what the physicians get paid. And the diagnosis is used for generating health statistics. So according to the traditional paradigm the diagnosis registered by the physician in the patient record could automatically be accessed by other systems. However, our preliminary analysis indicates that the physicians quite often use different terms to denote the diagnosis in the different documents. According to the traditional data modeling approach this is either an error or a mystery. According to Rommetveit's approach it can be explained by the context dependency of data. A physician will naturally use one term to denote the diagnosis in the medical record. That term will later be read by the physician together with other parts of the record next time she is visited by the patient. And the physician will interpret the term against the background of her knowledge about the patient and her tasks in which she will use this information. The context in which the diagnostic information will be used at a social security office is quite different, and the same data will accordingly be interpreted differently.

This problem can further be illustrated by the following example: At the University Hospital of Rennes in France they have, for several years, had a system containing Medical Record Summaries. The summaries are used automatically for several different purposes. This system has caused the establishment of a new medical profession, medical information archivists, who produce the summaries in cooperation with the medical staff. This new profession was established due to the knowledge needed to register the summaries correctly with regards as to how they would be used by different systems in different institutional settings (Le Beux 1990).

5.4 Conclusion

If we are ever going to succeed in our integration efforts, we need a thorough understanding of the problems we will confront. We make the task impossible if we limit our search for knowledge that can help us to narrow technical topics (or to problems related to use). Dahlbom says that "the remedy is not to be found in a 'new paradigm' but in a more informed use of the traditional paradigm." (Dahlbom 1988, p. 2/C 41) It is correct that we still need the power of the traditional paradigm. But we certainly need a new one as well — not to replace the old but to *embed* it! The informed use of the old one is, in my opinion, just to use it within the perspectives of a new dialectical-interpretive one.The traditional paradigm is, in short, based on the assumption that the world is a machine.This is a necessary assumption for making machines. But it is also an assumption with certain limitations. Standing within this ontology we are of course blind for these limitations. In this perspective, there is no task which cannot be taken over by machines. The only problem is that some machines become very complex. If our system does not work, we have built a machine with an error.

I find the most important problems related to software development to be caused by relying solely on this machine perspective. What we need is a complementary perspective which can help us to understand under what conditions the machine perspective can work, i.e. what kind of machines we should build as well as what kind we should not, and help us to understand why our machines do not work, i.e. when our hypotheses are not supported. A dialectical perspective appears to me to be such a complementary one. I think therefore think that it is a very worthy task to try to establish a new dialectical approach as many have argued for — in which, however, the old technical-rational one should be embedded.

6 Conclusion

In this paper I have argued that there will, in the future, be a huge demand for computer scientists with a thorough knowledge about organizations. Knowledge from both philosophy and the social sciences will be important. Industrialization of software development, more strategic use of software, and more integrated information systems are important trends causing this demand. Accordingly, I find the systems development educations in Scandinavia highly relevant. Dahlbom is right, however, when he points to the change towards more diversified use of computer technology (Dahlbom 1990b, p. 95). And he is also right in accusing the Scandinavian systems development community for not being aware of this. I also agree with Dahlbom in saying that technological based world views are necessary tools for generating ideas and visions about use of computer systems.

Acknowledgments

I would like to thank for all helpful comments and suggestions given by Bo Dahlbom. Thanks also to the members of my working group at the 13th IRIS conference in Turku, Finland 13 - 15 Aug. 1990, the anonymous reviewers, Jens Kaasboll, and to Eric Monteiro for their comments on earlier drafts of this paper.

Notes

1. As the arguments I will discuss here are presented in three papers by Bo Dahlbom, one of which written together with Lars-Erik Janlert, for reasons of simplicity I will refer to all these arguments as Dahlbom's.
2. This quote refers to a quote of Hegel" "Electricity is the purpose of the form from which it emancipates itself, it is the form that is just about to overcome its own indifference;" (Dahlbom 1990b, p. 86).
3. Examples of such product categories are data bases, operating systems, compilers, accounting systems, office systems.
4. ODA is a standard approved by ISO and CCITT as a standard for exchange of multi media documents (ODA).
5. X.400 is the joint ISO and CCITT standard for electronic mail.
6. SQL is the industri standard for relational data bases.
7. The technical problems are described in more detail in (Hannemyr 1989).

8. This system is under development by the Norwegian companies InfoMedica and Fearnley Data Services.

9. This prototype was developed as a part of the work for defining an international standard for exchange of medical information (Hannemyt 1990).

10. These are: (1) information technology is so unbelievably powerful, (2) we see an opportunity to politicize science, (3) in times of industrialization we come up with new, less easily industrialized, tasks for systems developers, and (4) we think that information technology is a very special sort of technology.

11. The topic of this section is more extensively analyzed in Hanseth 1991).

References

Le Beux, P., A. Burgin, C. Riou & P. Lenoir, (1990). An Integrated System of Medical Record. In: E. Dimitz, editor. *Proceedings from 'Computers in Hospital Care'.* June 8-9 1990, Vienna. Austria. p. 80–87.

Benjamin, R. I. & M. S. Scott Morton, (1988). Information Technology, Integration, and Organizational Change. *Interfaces.* 18(3).

Birnbaum, J. S., (1985): Towards the Domestication of Microelectronics. *Computer.* (November).

Carter, K. & A. Henderson, (1990). Tailoring Culture. *Preceding of the 13th IRIS.* August 12-15 1990, Turku, Finland.

Cox, B. J., (199). Planning the Software Industrial Revolution. *IEEE Software.* 7(6).

Curtis, B., H. Krasner & N. Iscoe, (1988). A Field Study of the Software Design Process for Large Systems. *Communications of the ACM.* 31(11):1268–1287.

Dahlbom, B., (1988). A New Paradigm for Software Development. In: Budde *et al.*, editors. *Software Development and Reality Construction.*

Dahlbom, B., (1990a). Using Technology to Understand Organization. In G. Bjerknes *et al.*, editors. *Organizational Competence in System Development. A Scandinavian Contribution.* Studentlitteratur, Lund, Sweden.

Dahlbom, B. & L.-E. Janlert, (1990b). An Artificial World. An Invitation to creative conversations on future use of computer technology. *Scandinavian Journal of Information Systems.* 2:85–100.

Dahlbom, B. & L. Mathiassen, (1990). Systems Development Philosophy. Paper presented at the 13th IRIS, August 12-15 1990, Turku, Finland.

Dreyfus, H. L., (1982). *What Computers Can't Do: A Critique of Artificial Reason.* Harper & Row, New York.

Dreyfus, H. L. & S. E. Dreyfus, (1986). *Mind over Machine.* Basil Blackwell, Oxford.

Dvergsdal, P., G. Hannemyr, O. Hanseth & H. Larsen, (1990). An ODA based system for standardized exchange of medical documents. To be presented at Medical Informatics Europe, Glasgow, August.

Ehn, P., (1988). *Work-Oriented Design of Computer Artifacts.* Almquist & Wiksell International, Stockholm.

Elster, J., (1983). *Explaining Technical Change. A Case Study in the Philosophy of Science.* Cambridge University Press.

Gilbert, G. N. & M. Mulkay, (1984). *Opening Pandora's Box. A sociological analysis of scientists' discourse.* Cambridge University Press.

Hannemyr, G., O. Hanseth & H. Larsen, (1989). ODA as a basis for standardization of exchange of medical information. A prototype system. Norwegian Informatics Conference, Stavanger, Norway.

Hanseth, O., A. Pape & M. Gritzmann, 1988). Egenutvikling eller kjop av standard-system. NCC-note ITIP/9.

Hanseth, O., (1991). Integrating information systems: The importance of contexts. To be presented at COSCIS '91, Helsinki, August.

Kaufmann, G., (1988). Problem Solving and Creativity. In: K. Grønhaug & G. Kaufmann, editors. *Innovation: A Cross-Disciplinary Perspective.* Norwegian University Press, Oslo, pages 87–138.

Larsen, H., editor, (1989). MIA - 3.2 - Multimedia Prototype System. Norwegian Computing Center and Norwegian Telecommunication Administration. Report no. 830, September.

Mathiassen, L., (1981). Systemutvikling og systemutviklingsmetode. DAIMI PB-136. Datalogisk afdeling, Matematisk institutt, Aarhus Universitet, 1981.

Morone, J., (1989). Strategic Use of Technology. *California Management Review.* 31(4).

Open Document Architecture (ODA) and Interchange Format, (1988). Introduction and General Principles, CCITT Recommendation t. 411, Melbourne.

Pfaffenberger, B., (1988). Fetishised Objects and Humanized Nature: Towards an Anthropology of Technology. *The Journal of the Royal Anthropological Institute.* 23.

Popper, K., (1989). *Conjectures and Refutations. The Growth of Scientific Knowledge.* Routledge, London, 5th edition.

Rommetveit, R., (). In search of a truly interdisciplinary semantics. A sermon on hopes of salvation from hereditary sins. *Journal of Semantics.* 2:1–28.

Rommetveit, R., (1986). Meaning, context and control. Convergent trends and controversial issues within current social scientific research on human cognition and communication. ESF Workshop, Zurich, September 1986.

Stamper, R., (1984). Management Epistemology. Garbage in, garbage out (and what about deontology and axiology ?) Paper presented on IFIP WG 8.3 Working Conference on Knowledge Representation and Decision Support Systems, Durham.

Weiss, A. R. & P. H. Birnbaum. Technological Infrastructure and the Implementation of Technological Strategies. *Management Science.* 35(8).

Winograd, T. & F. Flores. *Understanding Computers and Cognition - A New Foundation for Design.* Ablex Publishing Corporation, Norwood, N. J.

Winograd, T., (1986). A language/action perspective on cooperative work. In: *Proceedings from Conference on Computer-Supported Cooperative Work.* Austin, Texas, December.

Winograd, T., (1987). Thinking machines: Can there be? Are we? Paper presented at the conference on 'Humans, Animals, and Machines: Boundaries and Projections,' at the Stanford Humanities Center in April.