Scandinavian Journal of Information Systems

Volume 4 | Issue 1 Article 4

1992

DIFFUSION OF CASE: An obstacle race?

Lone Malmborg
Copenhagen Business School, LoneMalmborg@emailaddressnotknown

Scandinavian Journal of Information Systems, Vol. 3, pp. 105–118, 1991

DIFFUSION OF CASE
An obstacle race?

LONE MALMBORG
Institute of Computer and Systems Sciences at Copenhagen Business School
Rosenørns Allé 31,4, DK-1970 Frederiksberg C, Denmark

Abstract

This paper provides a conceptual and discussion framework for answering what the obstacles to diffusion of CASE-technology may be. CASE is viewed in a tool-external context and with focus on the negative aspects of CASE-technology, as the goal is to discuss the obstacles to diffusion of CASE. The paper is mainly based on literature reviews and serves as a point of departure for further empirical research.

Application of CASE-tools has consequences for the individual work process of the systems developers and for the rôles of the development team members. These consequences may cause resistance against the use of CASE-tools. Successful use of CASE-tools is also determined by the basic approach to development of software. Moreover CASE-tools influence the communication between developers and end-users. Increased quality of design and productivity of the development process depend on correctly understanding of the requirements, which reflects the quality of this communication.

An important condition of diffusion is solving of the so-called software crisis: lack of productivity and quality increases of software development. Productivity and quality increases necessitate efforts in the initial development phases and better support of the user-developer communication. These are areas, which are not seriously improved by the application of CASE-tools.

Keywords: Computer-Aided Software Engineering (CASE), systems development, software engineering.

1 Introduction

Computer Aided Software Engineering (CASE) has been introduced to the market as a means to solve the so-called software crisis. This crisis is exemplified by lack of productivity increases, missing attainment of quality gains and insufficient improvement of maintenance strategies. These factors are part of the reason why costs of software development and maintenance have been increasing compared to the overall costs of computer systems (Charette 1987).

There is no doubt that CASE-technology has strong advantages compared to manual techniques. Most important through the automation of tedious, time consuming documentation activities, some programming and configuration management activities. Colloquially the term CASE is used as a designation for very different things, including diagraming tools with no check-facilities as well as separate code-generators. In this paper CASE is defined from the main criteria, that the CASE-tool includes a repository, as suggested by Gane (1990): "the distinguishing characteristic of a CASE product is that it builds within itself a design database, at a higher level than code statements or physical data elements definitions."

Despite the above mentioned promises, efforts by both vendors and researchers have generally failed to validate the impacts of CASE-tools. According to Kemerer (1989) this problem stems from a number of difficulties in the CASE evaluation research problem. He expresses the possibility that CASE may be just another unproved productivity nostrum.

Many evaluations and classifications of CASE-tools are based on technical aspects of the CASE-technology. This rests mainly on the traditions for evaluation of software engineering environments (Firth *et al.* 1987, Charette 1987). By using these aspects, you evaluate the CASE-tools by focusing on how well the activities of the system life cycle phases are carried out. The basic approach to development of software embedded in CASE-tools is not questioned when the evaluation of the technology is based mainly on technical aspects, or put in another way, the CASE-technology is evaluated on its own premises by focusing on tool-internal aspects.

In the remainder of this paper I will discuss what the obstacles for the diffusion of CASE-tools—and with that software engineering principles—could be. I will focus on some aspects from a context external to tools. This means that the CASE-technology is discussed and evaluated not only on its own premises, but also by viewing CASE in a context external to tools. I will primarily focus on the negative aspects of CASE-technology as my goal is to make a conceptual discussion of the obstacles to diffusion of CASE-technology.

What is the so-called software crisis, and which influence will use of CASE-tools and use of engineering principles have on this? An outline of this situation will be presented in Section 2.

In Section 3 I look at probable obstacles from a sociological perspective. This

perspective is divided further into two sub-perspectives, one looking at individual factors, and one looking at the group or organizational factors.

Obstacles for a successful diffusion of software engineering principles through application of CASE-tools may be a question of a basic approach used in development of software. One approach is primarily concerned with problems related to the construction of the very system. Another is also concerned with problems and consequences related to the basis and environment of the system. This is discussed in Section 4.

In Section 5 I add proposals for further research and some concluding remarks.

2 Influence on the Software Crisis

The use of CASE-tools supports the methods used in the software *engineering* process. An engineering approach means disciplined application of scientific knowledge to resolve problems of immediate, practical significance. The engineering discipline relies on codifying scientific knowledge about a specific technological problem domain, in a form that is directly useful to the practitioner, thereby providing answers for questions that often occur in practice. When the practitioner uses engineering problem solving principles he does not work as a virtuoso, but applies earlier obtained—and operationalized—knowledge and insight. In that way greater efficiency, standardization, and measurability is obtained.[1]

2.1 A Historical Perspective on Software Engineering

Consider development of software as an engineering discipline Shaw claims: "In current software practice, knowledge about techniques that work is not shared effectively with later projects, nor is there a large body of knowledge about development of software organized for ready reference. Computer science has contributed some relevant theory, but practice proceeds largely independently of this organized knowledge" (Shaw 1990, p. 16). This claim leads me to question why the engineering approach is not more diffused through the area of software development, as the use of engineering principles through application of CASE-tools is seen as a means to solve the software crisis.

When introducing the waterfall model, Boehm (1976) proposed a definition of the concept software engineering: "the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them." This definition is consistent with the traditional definition of engineering, although Boehm was aware of the lack of sufficient scientific technological premises. Due to Boehm's insights, a lot of attempts to establish systematic procedures to support design and analysis activities in the software life cycle, were introduced. The work of Yourdon & Constantine (1979) and DeMarco (1978) probably exerted the greatest influence on these attempts.

Not until the emergence of CASE-technology in the mid eighties, was an attempt made to provide a complete and integrated computer support of software development. Scientific knowledge, economic rationality and established procedures are combined in tools which support (and partly automate) all activities of the entire software life cycle. Now it would seem that the basis for an engineering approach would seem to exist.

2.2 Improvement of Productivity

Findings in an empirical measure on the productivity of structured methods reported in Cerveny & Joseph (1988) suggest that structured techniques are more labour intensive than those non-structured techniques that are assumed to be inferior. However Cerveny & Joseph (1988, p. 250) concludes that "structured techniques still hold a promise for more productive systems analysis and programming. The development of CASE technology may dramatically change the productivity of these methods, by automating the tedious, time consuming documentation activities associated with structured analysis, as well as by eliminating the need for programming support."

Despite these questionable effects on productivity,[2] purchase and use of CASE-technology to support development of software is most often motivated by concerns regarding increasing and measuring productivity in development of computer systems, although there seem to be only a few indications of immediate productivity increases (Kemerer 1989), (Aaen & Sørensen 1991a). Productivity may even decrease during the introduction process as a result of the learning curve involved in starting to use the CASE-tool (Aaen & Sørensen 1991a).

It seems that with the application of CASE-tools, as with the application of structured techniques, the advantages are mainly in the maintenance costs due to a reduction of errors in the coding phase and to a higher degree of standardization. But these are not the most essential reasons for lack of productivity and management of costs in software development projects.

2.3 Communication Between End-users and Developers

Despite the relatively large amount of maintenance costs due to errors in the coding phase, most maintenance costs are caused by changes in the original requirements specification (Charette 1987). In (Yourdon 1989, p. 466) it says that there exists "an on-going misunderstanding between the end user and the systems analyst... This is important, because we know that 50% of the errors in a typical systems development project today are due to misunderstandings between the end user and the systems analyst; 75% of the cost of error removal in an operational system is associated with errors that originated in the systems analysis phase."

These statements indicate that the most essential problem in relation to development of computer systems is not at all affected by the application of CASE-

tools. When CASE-tools do not grasp the nettle, it is because they do not support the aspect of the development process that contains the embryo for the difficulties in management of costs and increasing of productivity: by the use of CASE-tools the user-developer communication in the initial phases and thereby the understanding of the users requirements are still haphazard and individually done. This question is explored later in Section 4.

2.4 Improved Communication Through Prototyping

However the CASE-tools may indirectly contribute to a solution of the above mentioned problem, because they can make prototyping possible, as they facilitate the rapid development of example systems. As a software development approach prototyping is essentially a technology-driven approach, which has matured through the appearance of CASE-tools. A prototype may in many situations constitute a basis for a much better communication between users and developers than usual diagrams from the structured methods may do. Ehn says that "prototypes can be used as alternatives or complements to traditional textual or graphic more or less formalized detached descriptions. Integrated with scenarios of future use, the experimental use of prototypes in design may be an important technique in playing the language-game of design, a game of involvement and by doing, that defeats some of the limits of formalization" (Ehn 1988, p. 206).

3 The Systems Developers' Work Processes

If a productivity increase is to be obtained by application of CASE-technology, it will also require a high degree of standardization. Because one of the most important aspects of CASE-tools is that they permit small parts of the system to be done by different people and integrated in the end.

In my opinion systems developers are not prepared to be governed by a very tight set of rules and procedures throughout the development process. The interesting question here is whether software developers will ever be ready for this way of organizing their work. The methodological basis for most CASE-tools is structured methods (Jarvenpaa *et al.* 1990). One of the most widely mentioned advantages of CASE is that CASE will automate much of the tedious work embodied in structured techniques.

But although structured methods are in widespread use, both theoretical criticism (Naur 1985) of the methods and empirical surveys (Bansler & Bødker 1989) indicate that the way in which systems developers use the tools of the method is quite individual. There is a wide gap between the way systems development is portrayed in the normative scientific and technical literature, and the way it is carried out in real life (Bansler & Bødker 1989).

Application of CASE-tools has consequences for both the individual work process of the systems developers as well as for the rôles of the development team members. If we view software development projects as projects of a very

unique character, it is not easy to apply earlier obtained—and operationalized—knowledge and insight. Tom DeMarco, who has influenced the software analysis and design area with standardized, operationalized techniques (and thus also has attempted to introduce an engineering practice) claims that a model for a software development project should be viewed "as a specification of one project. A methodology, by analogy, is a specification of all projects. The notion that there could ever be a single general-purpose specification of all projects is about as credible as the idea of writing a single general-purpose specification for all systems" (DeMarco 1982, p. 131). General guidance and common rules for the organization of the systems developers' work is not playing the most important rôle: Intuition, individualism, and autonomy are central values for the systems developers.

3.1 The Individual 'Theory Building' of Systems Developers

With CASE-technology the stage seems set for standardization of both product (the goal system) and process (the systems development process from generation of ideas to use of the system). It was a question of the same matter when the software life cycle approach (process) and structured methods (process and product) were introduced in the end of the 70's. But how can we explain that this standardization has not taken place despite the support from methods and tools.

Naur (1985) provides one possible theoretical answer to why systems development cannot be subjugated formalisms and methodological rules. Under the title *Programming as Theory Building* he argues in favour of the fact that the essential activity of the systems development process[3] is to "build a theory" of the actual domain—irrespective of whether the development process is conducted by individuals or a collective. Naur uses the word 'theory' in the sense of Gilbert Ryle[4] to denote something like understanding, enabling us to give qualified answers to questions pertaining to specific domain of interest, to act intelligently in relevant situations, and to take decisions concerning our actions. Theory is not simply taken to mean developing or having a command of abstract theorems on some complex domain. On the contrary: Theory building, in Ryle's sense also takes place in everyday situations. Also one can be said to have a theory when one is in a position to relate abstract knowledge to situations encountered in real life and to apply it fruitfully.

The guidance and control implied in structured methods and CASE-tools based on structured methods do not support the *theory building* and problem understanding that Naur advocates. Naur stresses as an essential part of this *theory building* the importance of the use of direct intuition in all areas; also in development and understanding of computer systems.

He considers it a fundamental mistake to underestimate the value of intuition, and to regard intuition as an unreliable factor. It should never, Naur continues, be replaced by a rule-based system of formalisms, such as the structured methods

and most CASE-tools. The more a process is controlled by rule-based systems, the less influence intuition will get. In CASE-tools, which follows a certain systems development method, there will be very little latitude for intuition.

In some CASE-tools, analysis of the application domain is supported in a way, such that the systems developer is offered a pre-completed matrix with standard business concepts, so the systems developer just checks off this matrix. In this way a standardization of the problem understanding is forced and the quality of individual, intuitive understanding is ignored.

Furthermore Naur claims that there will never be one right method for systems development in terms of theory building, and that there exist no formal methods or sequences of method steps which can be defined definitively in advance and be utilized for systems development.

Although Naur's idea about *programming as theory building* is influential and useful in relation to the understanding of how systems developers work with CASE-tools, I must state some critical objections. One critical objection is, that systems developers will intend to be very rational (and idealistic) in their approach, whereas the average developer is as best average (Parnas & Clements 1985). As a second objection to Naur's claim it could be stated that individual experience and intuition might co-exist with formal methods. Especially when the developer is a novice in his area, some degree of formalization as a kind of guidance might be helpful in performance of complicated systems development activities.

However there seem to be some empirical evidence, that (at least in Denmark) systems developers work in a more individual way. In a survey of systems developer values in Denmark and Canada (Kumar & Bjørn-Andersen 1990), it was found that strong values to the Danish systems developers meant autonomy in planning and performance of tasks, while the Canadian developers put stronger emphasis on reliability and operating costs. Another survey (Hørlück 1985) points in the same direction. Hørlück's survey reports about a low degree of formalization in software development in Danish enterprises compared with software industries in two other industrialised countries (Holland and The United Kingdom).

Both of these surveys indicate that a more flexible way of solving problems takes place in Denmark (and Scandinavia) than in the other industrialized countries. This fact points in the direction that application of engineering principles in software development, would have difficulties establishing in Denmark (Scandinavia). This indicates that Danish design values and traditions will not favour the adoption and use of CASE-tools. It seems that informal work procedures have a strong position in Denmark as well as that there is a widespread resistance against the increased formalization which the use of structured methods implies.

This resistance can presumably be traced back to Naur's claim that *programming is theory building* where *theory building* in this context is to be understood as the individual unfolding of creativity. The programmer's skill and intuition is

the pivot upon which the activity turns, and the use of specific techniques, tools and rules plays a minor, secondary rôle.

3.2 Method Guidance in CASE-Tools

Naur's assumptions about theory building in the context of systems development and the two empirically based surveys point in the direction of two matters: first it seems that the very nature of the design process puts obstacles in the way of a completion of a professional engineering approach. Because this fact must mean, at least for Scandinavian systems developers, that a successful adoption and use of CASE-tools depend on whether the CASE-tools are flexible enough with respect to methods and leave room for individual creativity and use of intuition in the design process.

One of the main figures behind a well-known CASE-tool was very promising about the company's CASE philosophy: "We wanted to build houses of worship and not religions back in 1984. We wanted to provide a set of tools within an environment. We weren't going to constrain people. We didn't want to be in the religion business" (Chris Grejtak, Index Technologies, in Jarvenpaa *et al.* (1990, p. 2)).

The major part of all CASE-tools follows the procedures and rules of structured methods, but an American survey (Jarvenpaa *et al.* 1990) of those PC-based CASE-tools that provide support of structured analysis and design, shows that there are clear differences in the flexibility of the tools. The survey categorizes the tools into three types in respect of method support: restrictive, guided, or flexible. The three categories can be described by an example:

The *restrictive* approach will force the developers to conduct a top-down decomposition of dataflow diagrams. The systems developers will not be able to choose to start constructing a dataflow on a lower level in the hierarchy. With a *guided* approach the systems developers will be prompted to use the above procedure. During the whole process the CASE-tool will give advices instead of directives. With a *flexible* approach the systems developers can choose by themselves which approach they want to use, and whether they want verifications of their procedures later on.

The method support can in the actual situations be of two different kinds: automated control or informative feedback on methodological violations.

A notable issue in (Jarvenpaa *et al.* 1990), which lies in continuation of Naur's idea of 'theory building,' is that although structured analysis results in a number of hierarchical specifications, design does not necessarily take place in a top-down way. According to Jarvenpaa *et al.* true experts rather behave opportunistically in responding to conditions as they arise during the design process. Experts tend to apply a top-down balanced decomposition approach when they are familiar with the problem, but work at different levels of abstraction when they are less familiar with the problem.

3.3 The Development Team

A different aspect of CASE-technology, which affects the systems developer's way of working, and which can constitute an obstacle to a widespread use of CASE-technology, is the changed division of labour that occurs in the development team when using CASE.

In other areas there is a historical evidence that application of engineering principles causes resistance among the involved groups. A specialization often means dequalification, because there will be areas of the profession, that there will no longer be a need for. It can either be because the technology automates manual procedures, or it can be because tasks are placed in the hands of other groups or professions. In the last case, it will possibly also cause a change in power relations.

Wanda Orlikowski has made a survey of the social implications of the use of CASE-tools. The survey is described in (Orlikowski 1988, 1989). The final conclusions of the findings are that the use of CASE-tools results in "structural changes due to modification of the systems development division of labour and shifts in patterns of dependency among project team coalitions. These changes triggered a polarization among the systems developers which was evinced in acts of coercion and rebellion, the display of territorialism, resentment, and stereotyping, as well as the enactment of subcultures" (Orlikowski 1989, p. 199). These structural changes might be the natural, short-term result of power shifts that follow most revolutionary technologies. In the long term this means that the resistance among the involved groups will disappear, when a new division of labour has existed for a while. A changed division of labour will probably only be a provisional obstacle for diffusion of CASE-tools.

4 The System and the Surroundings

The point of viem that the engineering approach to software development is too narrow in its approach to development of information systems could be another possible obstacle for widespread use of CASE-tools.

This problem is reflected in the relationship between the fields of (or approaches implied in) software engineering and systems development. This distinction between software engineering and systems development is made by Bansler & Clausen (1989). The latter is also sometimes denoted information systems development, software development or computer systems development. The two fields have many points of contact: project management, introduction of methods and tools for supporting the development of software etc. But the subjects of the two areas are nevertheless distinctively different from each other.

With an engineering approach to software development we are by convention primarily concerned with problems related to the construction of the very system and not with problems and consequences related to the basis and environment of the system (Bansler & Clausen 1989). We should carefully distinguish devel-

opment projects that are commenced from a computer systems viewpoint, from those that are commenced from a computer-based systems viewpoint. The distinction is subtle, but critical in understanding the developer's frame of mind. While software engineering typically is concerned about computer systems, systems development typically is concerned about computer-based systems.

Taking computer systems as a starting point implies that you are interested in constructing computer systems. A usual assumption about this approach to software development is that a clear and definite description of the problem is given in advance. Most likely, you are often only concerned about the technical problems that occur when the system is implemented. By taking this approach to software development, you remove the problem which is often seen as the most important reason for lack of productivity and bad management of costs in software development. This problem concerns changes in the original requirement specifications. Such changes can be traced back to the misunderstanding of the user organization's problems and needs. These changes are also related to the poor communication and misunderstanding between users (or user organizations) and developers.

Taking computer-based systems as a starting point implies that your are also interested in the problems and changes that introduction and use of computer systems in organizations will cause. With this starting point the development process is also a process of getting the users to understand, formulate, and define their problems and needs, instead of letting the managers or consultants formulate a number of fixed problems—and requirements—as input to the developers: "A primary goal of CASE products is to automate much of the clerical work involved in applying the techniques, so that the analyst can devote more time to understanding the user's requirements and produce a more complete and consistent specification" (Lejderman 1987, p. 80).

4.1 Problem Solving or Problem Definition

If the method built into the CASE-tool views the design process as a problem solving process—where the problems are defined and presented in advance by the managers—and not as a problem definition process, and if the CASE-tool is guided or restrictive (in Jarvenpaa's terms) with regard to this method it cannot be assumed that the use of CASE-tools automatically means improved understanding of the users' need. The reason for this is that the most important argument for managers of software development projects to buy CASE-tools is reduction of costs in the entire development process. Thus, the time saved is not used for better communication with end users, but to increase productivity.

CASE-tools appear to support the construction of computer systems and not the understanding of the problems and changes the system will cause for the user organization. Indeed, CASE-tools can put more focus on the technical aspects of systems development, because most CASE-tools take for granted that the

problem to be solved is well structured and defined. Thus, the design process is viewed as a problem solving activity instead of as a problem definition process.

Malhotra *et al.* (1980) proposes that instead of viewing the design process as a problem solving process, we split the process into three iterative processes: goal elaboration, design generation and design evaluation, because in "real-world design situations, the goals are, typically, fuzzy and poorly articulated and cannot be mapped directly into properties of the design. Thus, exact configuration of the final state is not prescribed. A part of the design process consists of formalizing and refining the design. Even so, it is usually difficult to tell how well a design meets a particular functional requirement. In addition, the functional requirements often cover different dimensions and the trade-offs between them are rarely well specified" (Malhotra *et al.* 1980, p. 120).

As mentioned before, CASE-tools take for granted that there exists a well-defined problem (and goal). They do not support this aspect of the systems development process. This partly constitutes the reason why CASE is an obstacle to management of costs and productivity. CASE-tools still give far too little priority to the communication with the users and a better understanding of the application domain and the user organization. The same can be said about the structured techniques on which most CASE-tools are based.

Some CASE-tools can be used in ways that get around the problem noted by enabling a rapid prototyping approach. In this way the prototype serves as a kind of specification, instead of the usual written specification. This prototype can constitute a basis for a better communication between users and developers than the traditional diagrams from the structured methods. The diagrams will in most cases be an unsatisfying basis for a discussion between users and developers about to which degree the specification meet the users' need. Unfortunately, it seems that prototyping is not applied to any greater extent with the introduction of CASE-tools (Cerveny & Joseph 1988, Aaen & Sørensen 1991).

I am convinced that the diagrams of the structured methods are supplied with additional authority, because they are presented as more perfect and completed. Thereby they will be less receptive for proposals for change, than the former handdrawn diagrams were. In my opinion, CASE-tools will—to a much higher degree than the structured methods alone—shift the user-developer communication to a mode of communication clearly founded on the developers' premises alone.

5 Concluding Remarks

In the preceding I sketched some obstacles for diffusion of CASE-tools.

I claimed that one important obstacle is that CASE-tools have negative influence both on the systems developers' individual way of performing their work as well as on the division of labour in the development team. To me the most important obstacle is that CASE will not have distinctive impacts on a very central

barrier for productivity: understanding the users' need by establishing a better communication between users and developers. Finally, I claimed that CASE need to support a problem definition process instead of a problem solving process.

But when these negative aspects of CASE are mentioned, I must stress that I believe that there is no doubt about the fact that CASE-technology has come to stay in software development. But rather than letting the CASE-tools guide the process of developing software, the CASE-tools should be part of the systems developer's tool box. This tool box can supply the systems developer's intuitive and creative activities in understanding the users' essential needs. And rather than a means to productivity increase, the CASE-tools may be viewed as a means to increase quality of both the process and the product: give the systems developer tools that can be integrated, that remove tedious working procedures, that help to document the process, that offer alternative ways of performing activities and useful advice, that render the possibility for several approaches to problem solving and for prototyping.

5.1 Further Research

As this paper is mainly based on literature reviews it should be followed up by some empirically based research on how system developers work with CASE-tools. Orlikowski's surveys are among the very few research projects of this sociological kind (Orlikowski 1988, 1989). Most research projects until today are focusing on technical aspects of the tools in order to categorize existing tools (Jarvenpaa *et al.* 1990), or are in a quantitative way focusing on diffusion of CASE-tools in terms of which kind of companies use the tools, on how many projects and for which kind of tasks (Aaen and Sørensen 1991b). The research in the area of CASE-technology need studies with focus on qualitative consequences for aspects from a context external to the tools such as communication between users and developers, problem definition and problem structuring, and division of labour among developers.

This paper has served as motivation for further work with my Ph.D. dissertation and expresses a stage in evolution of my own thinking. Since the first version of this paper was written I have carried out a project (B4CASE[5]), in which it was studied how user-developer communication and the problem definition process could be improved. As part of the project we developed a front-end for CASE-tools. The aim of this front-end was to support the problem structuring process preceding the traditional analysis phase and communication in general in systems development.

Acknowledgements

I would like to thank Carsten Sørensen, Pasi Kuvaja, Liam Bannon and the reviewers of Scandinavian Journal of Information Systems for useful comments on earlier versions of this paper.

Notes

1. This definition of the engineering discipline is inspired by Shaw (1990).
2. It can be very difficult to measure the effects in terms of increased productivity, as discussed in (Kemerer 1989). The most commonly used productivity metric, according to Kemerer, is source lines of code (SLOC) per labour unit. This productivity metric suffers from two distinct sets of problems. First there is a general difficulty in capturing comparable SLOC metrics, and second there is a problem getting agreement on whether increasing this ratio is desired, and how it can be used to assess new CASE tools that directly affect the numerator by changing the language level or by automatically creating the SLOC, as in code generators.
3. In Peter Naur's paper *Programming as Theory Building* he does not use the term 'systems development.' Instead he is talking about 'programming', here not in the sense of 'coding' as it most often is used. When he uses the word 'programming', he denotes "the whole activity of design and implementation of programmed solutions. What I am concerned with is the activity of matching some significant part and aspect of an activity in the real world to the formal symbol manipulation that can be done by a program running on a computer. With such a notion follows directly that the programming activity I am talking about must include the development in time corresponding to the changes taking place in the real world activity being matched by the program execution, in other words program modifications" (Naur 1985, p. 253).
4. Naur uses the work of the English philosopher, Gilbert Ryle: *The Concept of Mind.*
5. The results of the project is reported in Malmborg & Pries-Heje (1991).

References

Bansler, J. & H. Clausen, (1989). *Fire perspektiver på systemudvikling.* DIKU-Report 89/15. Institute of Computer Science, Copenhagen University.

Bansler, J. & K. Bødker, (1989). *A Reappraisal of Structured Analysis.* Roskilde University.

Boehm, B. W., (1976). Software Engineering. *IEEE Transactions on computers*, C-25(12):1226–1241.

Cerveny, R. P. & D. A. Joseph, (1988). Effects of Software Engineering on Productivity. *Information & Management*, (14):243–251.

Charette, R. N., (1987). *Software Engineering Environments. Concepts and Technology.* McGraw-Hill, New York.

DeMarco, T., (1978). *Structured Analysis and System Specification.* New York.

DeMarco, T., (1982). *Controlling Software Projects.* Yourdon Press, New York.

Ehn, P., (1988). *Work-Oriented Design of Computer Artifacts.* Arbetslivscentrum, Stockholm.

Firth, R., V. Mosley, R. Pethia, L. Roberts & W. Wood, (1987). *A Guide to the Classification and Assessment of Software Engineering Tools.* Technical Report, Software Engineering Institute, Carnegie-Mellon University.

Gane, C., (1990). *Computer-Aided Software Engineering—The Methodologies, the Product, and the Future.* Prentice-Hall, Great Britain.

Hørlück, J., (1985). A study of the Actual Usage of Methods and Techniques and a Discussion of their Productivity. In M. Lassen & L. Mathiassen, editors, *Report of the Eighth Scandinavian Research Seminar on Systems Engineering*, Part I. Dept. of Computer Science, Aarhus University. Pages 100–116.

Jarvenpaa, S. L., N. Tractinsky & I. Vessey, (1990). *Evaluation of Vendor Products: CASE Tools as Methodology Companions.* University of Texas at Austin.

Kemerer, C. F., (1989). An Agenda for Research in the Managerial Evaluation of Computer-Aided Software Engineering (CASE) Tool Impacts. *Proceedings of the 22nd Hawaii International Conference on System Sciences.* Pages 219–228.

Kumar, K. & N. Bjørn-Andersen, (1990). A Cross-Cultural Comparison of IS Designer Values. *Communications of the ACM*, 33(5):528–538.

Lejderman, J., (1987). A Look at the Case for CASE. *Canadian Data Systems*, 6:80–81.

Malhotra, A., J. C. Thomas, J. M. Carroll & L. A. Miller, (1980). Cognitive Processes in Design. *International Journal of Man-Machine Studies*, 12:119–140.

Malmborg, L. & J. Pries-Heje, (1991). *B4CASE. En metode og et værktøj til problemstrukturering i systemudvikling.* Working paper. Institute of Computer and Systems Sciences, Copenhagen Business School.

Naur, P., (1985). Programming as Theory Building. *Microprocessing and Microprogramming*, 15:253–261.

Orlikowski, W. J., (1988). CASE Tools and the IS Workplace. *Proceedings of the 1988 ACM SIGCPR Conference on the Management of Information Systems Personnel*, April 7-8, College Park, Maryland. Pages 88–97.

Orlikowski, W. J., (1989). Division Among the Ranks: The Social Implications of CASE Tools for System Developers. *Proceedings of the Tenth International Conference on Information Systems*, December 4-6, Boston. Pages 199–210.

Parnas, D. L. & P. C. Clements, (1985). A Rational Design Process: How and Why to Fake it. In G. Goos & J. Hartmanis, editors. *Lecture Notes in Computer Science*, No. 186, Springer-Verlag, Berlin. Pages 81–100.

Shaw, M., (1990). Prospects for an Engineering Discipline of Software, *IEEE Software*, 11:15–24.

Yourdon, E. & L. L. Constantine, (1979). *Structured Design.* Yourdon Press/Prentice-Hall, Englewood Cliffs.

Yourdon, E., (1989). *Modern Structured Analysis.* Prentice-Hall, Englewood Cliffs.

Aaen, I. & C. Sørensen, (1990). *The TEQ-project—Tools for Efficiency and Quality.* A project outline. Draft, Aalborg University.

Aaen, I. & C. Sørensen, (1991a). A CASE of Great Expectations. *Scandinavian Journal of Information Systems*, Vol. 3:3–23.

Aaen, I. & C. Sørensen, (1991b). *CASE i Praktisk Brug—Resultater fra CASE Monitor Projektet.* Aalborg University.