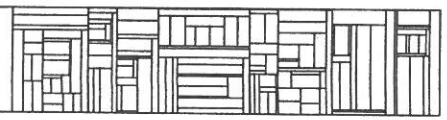
ISBN 87-88715-11-6
ISSN 0105-8517

FORMALIZATIONS IN SYSTEMS DEVELOPMENT

Lars Mathiassen
Andreas Munk-Madsen

DAIMI PB-193

Computer Science Department
AARHUS UNIVERSITY
Ny Munkegade – DK 8000 Aarhus C – DENMARK
Telephone: 06 – 12 83 55



MARS-report No. 11, July 1985

TRYK: DAIMI/RECAU

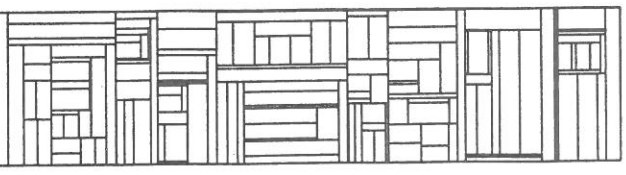
FORMALIZATIONS IN SYSTEMS DEVELOPMENT

by

Lars Mathiassen and Andreas Munk-Madsen
Computer Science Department
University of Aarhus
Denmark

DAIMI PB-193

Computer Science Department
AARHUS UNIVERSITY
Ny Munkegade – DK 8000 Aarhus C – DENMARK
Telephone: 06 – 12 83 55



MARS-report No. 11, July 1985

Abstract

Formalizations are related both to types of expression and to types of behaviour. The limits to applying formalizations in these two senses are discussed and illustrated by examples from practical systems development. It will be established that formalizations are valuable in some situations, but insufficient in others. The alternative to uncritically using formalizations is that system developers analyse the situations in which they find themselves, and from there plan a combination of a formal and an informal approach.

Note: This paper has been published in a shorter version in Ehrig, H. et al. (eds.): Formal Methods and Software Development. Proceedings of the International Joint Conference on Theory and Practice of Software Development, Berlin, March 1985. Springer-Verlag 1985.

1. Introduction 1

2. Methods for Formalization 1

3. Description 4
   3.1. Possibilities and Problems 4
   3.2. Limits to Formalization 8
   3.3. Alternatives 11

4. Management 11
   4.1. Possibilities and Problems 12
   4.2. Limits to Formalization 15
   4.3. Alternatives 18

5. Summary 19

References

1. Introduction

For several years computer scientists have enganged in discussions on methods for program development. These discussions have not always left practitioners with clear advice on and guidelines for programming. On the contrary, a major aspect like the significance of formalizations has provoked clearly conflicting viewpoints.

Our interest lies with methods for developing computer based systems in organizations. Program development is thus only a sub-activity in the wide spectrum of activities with which we are concerned. These activities are analysis, design, coding, testing, conversion; and not to forget, the very important activities of systems development management.

We experience a wide gap between the discussions in scientific literature and practical systems development. In spite of the intensity of the discussions formalizations are only used to a limited extent. It is natural to pose the question: Why? - What are formalizations after all? What are the limits to using formalizations in systems development? Which alternatives to formalizations can be proposed? This paper will discuss these questions. Examples of situations in practical systems development will be used as illustrations.

2. Methods for Formalization

In our terminology a *method* consists of prescriptions for performing a certain type of working process (Mathiassen 81). In addition to these prescriptions, a method is characterized by its *application area* - i.e.

the type of working processes in which the method may be applied - and its *perspective* (i.e. some assumptions) on the nature of these working processes and their environment.

The prescriptions of a method are given in terms of: techniques, tools, and principles of organization. A *technique* is a way of performing a working process with regard to the nature of the task and product. A systems development method may, for instance, include stepwise refinement as a programming technique. A *tool* enters into the working process as an aid. Usually at least one technique is attached to each tool. Structure diagrams (Jackson 83) are an example of a description tool prescribed in a systems development method. *Principles of organization* prescribe how the work should be performed under the given conditions. Conditions include the fact that resources are limited, and the fact that several people have to cooperate. Dividing a project into phases with built-in checkpoints is an example of applying a principle of organizing a systems development process. This principle serves to improve the control of the process.

In our context - systems development - the term formal may be connected to types of *expression* (descriptions, specifications, programs), and to types of *behaviour* (when carrying out systems development and when programming). According to Oxford Advanced Learner's Dictionary of Current English, *formal* denotes "in accordance with rules, customs, and convention". In the more restricted context of program development, Naur understands the term formal in the specific sense of: expressed purely by means of symbols given a specialized meaning. Furthermore he stresses that the formal mode of expression merely is an extension of the informal

one, not a replacement of it (Naur 82). We agree with this view, and consequently talk about degrees of formalization.

Seen from the point of view of formalization a method for systems development can provide at least two interesting types of prescription. First it can prescribe the use of description tools and related techniques which implies a certain degree of *formalized expression*. Secondly it can prescribe the use of principles for organizing the work which implies a certain degree of *formalized behaviour*. See figur 1.

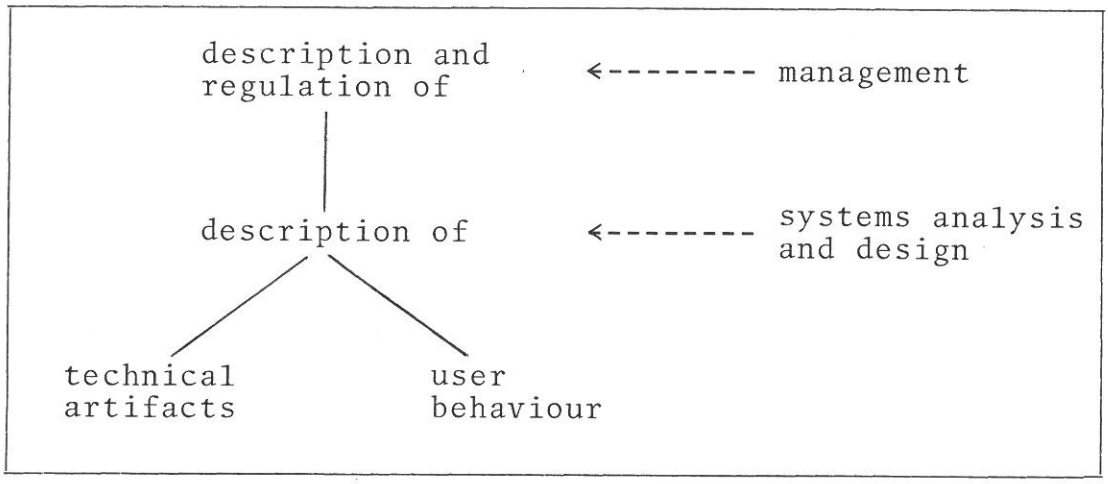


Fig. 1. Formalization relates both to the level of systems analysis and design, and to the level of systems development management

Many discussion can be traced back to the fact that this distinction has not been made clear. Method designers usually create the first type of guidelines and call these methods; software managers think they buy the second type and are badly disappointed.

In the following we will discuss the limits and alternatives to formalizations. Regarding the limits we are primarily concerned with *when* formalizations are useful

(application area). We give less attention to the question of the degree of formalization. Section 3 will discuss formalizations in relation to the use of description tools and the techniques attached to them. Here we address the issue of description. Section 4 will discuss prescriptions for formalized behaviour, especially principles for organizing development activities. Here we address the issue of management.

3. Description

Descriptions - of any degree of formalization - play an important rôle in the systems development process. One of the most important sub-products - the program code - is a formalized description. Descriptions of computer systems and the users' work and organization appear in all activities which directly aim at manufacturing the product: analysis, design, coding, test, and conversion. Important intermediate products include: descriptions of the users' current work and organization, functional requirement specifications, overall technical design, overall functional design, detailed technical design, detailed functional design, code, technical conversion plan, and functional conversion plan.

3.1. Possibilities and Problems

Example 1.

A system development project aimed at developing an interactive budget system. The overall design of the new system had been reviewed and accepted. One of the next steps was to design a module which would accept statements of amounts in various currencies from a character string. This module should recognize valid inputs and transform them into an internal representation, and it should give appropriate messages.

In this case formalization of the set of valid inputs was helpful. The programmer chose to specify a table, describing a finite state machine accepting valid inputs. By doing so the programmer obtained several advantages. The correctness of the specification was in this case intuitively clear, and the program structure could be derived almost directly from the input description.

Naur mentions tabular descriptions as an example of profitable application of formalization (Naur 82). Tabular descriptions are, as he puts it, the obvious means for helping to ensure that in a certain situation all cases, or any combination of cases, are considered and treated properly. More generally Naur argues that simple formalizations are of great practical value. Any of the various descriptions which are created during a system development effort may in fact employ any number of different formal notations side by side without contradictions. Using simple formalizations can both be practical and efficient.

Regarding descriptions in general we see at least four motivations for formalizations:

1. Formalized descriptions are imperative in the man-machine dialogue, because machines so far only can interpret and execute formalized descriptions.

2. A formalized description can be an effective means to avoid ambiguity and obtain conciseness in the communication between people.

3. A required use of formalizations can support the system developer's understanding because they force him or her to think.

4. In systems development several types of descriptions occur, including requirements and designs. If the problem can be described in a formalized way, the solutions may be more easily deduced, or it may be easier to verify the solutions.

In example 1 the third and fourth of these points are met. In our experience this will be the case in many system development situations. We find that practitioners are too poorly acquainted with the various tools for formalizing descriptions. Michael Jackson's structure diagrams (Jackson 75) are commonly known, but only a minority know of the existence of an equivalent notation: regular expressions, which are linear and therefore both convenient and effective.

Example 2.

During the development of a computer based production planning system much time was spent trying to specify the computation of throughput time. Everyone involved knew the meaning of this quantity - throughput time was in this connection a well-known term. However, every suggestion for a formalized specification of a calculation procedure was met with the same critisism by the production planners: the suggested calculation was too simple. Finally it became clear that it was not a case of specifying an existing computation procedure. The production planners had never before computed the throughput time - the values were estimated on the basis of personal experience, simple individual principles, and knowledge of the given situation.

Example 2 indicates that the system developers took the wrong approach because they wished to employ a specific tool. It would certainly be nice if the computation of the throughput time could have been specified in a formalized way - because then the assignment would have been more or less completed. But the formalized approach was not suitable in this situation. The problem was not to specify the computation of throughput

time, but rather to analyse and design how reasonable estimates of throughput time could be determined with the aid of computer based tools. The alternative would be to start analysing the production planners' work, and how the throughput time occured in their work.

Part of literature sees program development as an activity which takes its starting point in a well-defined situation where the problem and the type of solution is known. In reality the situation is rarely well-defined from the start. As Polya puts it in terms of practical problem solving in general: "unknowns, data and conditions are more complex and less sharply defined in a practical problem than in a mathematical problem." (Polya 57). As implied in example 2, one of the fundamental issues in systems development is in fact to understand the situation in terms of problems and possible solutions, e.g. to determine what the system should be able to do, and how it should interact with the organization's working processes.

Example 3.

During the development of an accounting system a problem surfaced as the design activities expanded. The system developers designed one solution after the other, but all solutions were rejected by the users, either on the grounds that the proposed system did not integrate the accounts of the hitherto separated company divisions, or on the grounds that the proposed system would radically change the way of working in one of the accounting departments.

The basic problem in this situation was that the system developers were faced with inconsistent requirements: on one hand their assignment was to develop a system which was common for all the departments, on the other each department wished to maintain their individuality. A naive suggestion would be that a formalized descrip-

tion of the requirements would have surfaced this problem earlier. But in the actual case the project group did realize that they faced inconsistent requirements. They just hoped that they could provoke a decision by working out and presenting various suggestions for the design of the system. They did not, however, succeed in this, and many efforts were wasted.

Example 3 illustrates one of the difficult problems in systems development. Inconsistent requirements appear frequently, and sometimes they are solved through open discussions. What really makes this case difficult is that the users in the different departments did not want to face the underlying conflict.

Any description tool would be of little help in this situation. The problem *appears* to be that the users are dissatisfied with a given design proposal; they want a more sophisticated design. Accepting this interpretation the system developers are left with going back home to specify a more refined solution. In principle the situation could be handled in this way, i.e. by making two systems in one. This solution would, however, neither be economical nor practical (it would involve account numbers with 40 digits). The point is that the situation *appears* to be a description or a design problem, but in *reality* it involves a latent conflict in the organization. This suggests that the system developers should force the organization to take a stance.

3.2. Limits to Formalization

We think that ideally the following conditions should be met before a tool for formalizing descriptions is applicable in a given situation:

1. The syntax and semantics of the tool should be well-defined.

2. The tool should be tested in practical situations.

3. A phenomenon which is suitable for formalized description must be identified.

4. The tool should fit the task. I.e., the tool should capture the characteristics of what is is to be described.

5. The system developers should be trained in using the tool.

Let us relate the examples to conditions 3 and 4. In example 1 both conditions were met, and the application of formalizations proved useful. In example 2 condition 3 was apparently met: the system developers found a phenomenon, i.e. computation of throughput time which was suitable for formalized description. However, focusing on this phenomenon represented a misinterpretation of the assignment. Here the application of a specific tool misled the system developers. In example 3 the system developers attempted to design several solutions in a situation where they faced conflicting requirements. What appeared to be a description problem was in reality an organizational problem. Here conditions 3 and 4 were in a way met, but no description tool would do in the situation.

More generally the state-of-the-art and the nature of systems development can be related to the above-mentioned conditions as follows:

1. There are many well-defined tools.

2. The simple tools have been tested. Method makers often promote the more advanced tools without drawing attention to the fact that the tools are still on the experimental stage.

3. Descriptions of computer systems can be formalized. Descriptions of organizations can only partly be formalized due to their social nature.

4. The application area and the perspective of description tools are seldom properly defined.

5. System developers in general know too little about tools for formalizing descriptions.

Limits to formalization may also be discussed by distinguishing between various types of description situations. Inspired by Lanzara (Lanzara 83) we may distinguish between the following situations:

- *routine* situations where both the problem and the type of solution is known,

- *problem solving* situations, where the problem is known, but the type of solution is unknown,

- *problem setting* situations, where the problem is unknown to the system developer.

In these terms example 1, 2 and 3 refer to routine, problem solving, and problem setting situations respectively. And our basic claim is that the more we move away from routine situations, the less applicable are the tools for formalizing descriptions, and the more profound is the need for alternative approaches.

3.3. Alternatives

The quality of the existing tools, seen in relation to the nature of the system development process, indicates that the system developers should have the possibility to *choose their own tool in any given situation*.

Formalizations are of great practical value as a possible extension of a basically informal means of expression. Today practical systems development is very much based on informal means of expression. The quality of the specifications might improve through a *more disciplined application of informal means of expression*, and through an *increased partial application of formalizations*. (Naur 82).

When attention is drawn to formalizations there is a tendency to neglect the activities in the system development process for which formalizations are unsuitable, especially the analysis of the users' work and organization. *Tools and techniques for analysing and designing the users' work and organization* ought to play a more dominant rôle in the discussions and in research.

Relating description issues to organizational problems and conflicts can be achieved through an *experimental strategy* (Floyd 84). An experimental strategy can be advantageously employed to clarify situations characterized by ambiguity and uncertainty (Davis 82).

4. Management

We now turn to the second interpretation of formalizations. This section will discuss formalizations - especially principles of organization - in connection

with system development management. The term management denotes all the activities in a system development project which are necessary because the project is carried out by more than one person, and which do not directly contribute to the production of the system. These management activities include planning and evaluation of resources, activities, and products; regulation of conditions; configuration management; and general management activities like marketing and motivation. Depending on the organization, some of these activities are performed by the system developers themselves, others are performed by managers.

The central question is: in which situations does system development management benefit from formalizations as prescribed in the rules and procedures of a method?

4.1. Possibilities and Problems

Example 4:

In a project the amount of work was estimated when the overall design was almost completed. The project was broken down into tasks which had an estimated size of 100 to 400 man hours each. The estimates were based on the system developers' rather extensive experience with the application and the development environment. However, system test and conversion were estimated to only 6% of the total development time. It actually took 20%, which is close to the textbook recommendations (Boehm 81). But neither textbooks nor company statistics were consulted.

Here formalizations could have helped to improve an important intermediate product - the estimate. The rule: "Compare the allocation of resources for activities with statistics" is easy to implement, and the advantages of doing so are self-evident.

Regarding formalization of behaviour in general we see at least two motivations:

1. Formalization is a means for increasing the quality and efficiency of working processes.

2. Formalizations is a means for increasing the efficiency of external control of working processes through reports and directives.

In example 4 the first of these points is met.

Example 5:

A project was heavily up-staffed after one man year had been spent on overall design. The design was, however, not completed at this stage. A further 15 man years were spent with rather chaotic programming activities.

The existence of a procedure for product acceptance, e.g. a "formal technical review" (Freedman and Weinberg 82) might have given management a warning not to up-staff. In this situation it is, however, more doubtful whether such a procedure would have been of any use. The first risk of failure is that the review offers an incorrect assessment. The second risk is that it might be decided that there is no time for a review because the project is behind schedule. And the third risk is that management might choose to ignore the review report. In the actual case management already acts at variance with general experience by violating Brooks' law: "Adding manpower to a late software project makes it later" (Brooks 82).

Example 6:

A project worked in an organization where procedures required that a steering committee accepted the overall design. The steering committee, however, accepted an overall design which had many defects. Confronted with critisism the project leader admitted the defects, but did nothing to improve the product, arguing that the product had been accepted. Later the introduction of the new system had to be postponed because of serious defects in the system.

Here a procedure is applied which is based in the second motivation mentioned above - but only ostensibly. This results in a wrong picture of the situation - which is really worse than a blurred one. The point is that rules and procedures can be used to place responsibilities formally. However, as we have seen in this example, rules and procedures may also work as pretexts for doing nothing, and they can support opportunistic adjustment of behaviour. They may thus - directly contrary to the intention - counteract genuine problem solving. This phenomenon is generally known as the dysfunctional effects of bureaucracies (March and Simon 58).

Example 7:

The method section of a data service organization was responsible for improving the working practices in systems development. The section saw its main task in producing guidelines which were adapted to the organization. Thus the activities mainly consisted of looking for solutions in the available literature, in participating in courses and meetings, and in writing. The results of these activities were mainly reports containing guidelines which were only observed to a modest degree in the organization.

The immediate problem is that the section is primarily concerned with producing guidelines, while it ignores the follow-up activities which should ensure that the guidelines are observed in practice. In relation to the chosen strategy for changing working practices, the section only solves one part of the task. The heavy problems related to bringing new methods into practical use are left to random initiatives. Why then is it carried out in this way? A simple answer would be that the method section is staffed with system developers, and that they think that programmers are programmable (Weinberg 82).

There is, however, an underlying problem connected to the chosen strategy. Changing working practices is separated as an independent function, and no attention is paid to the actual problems related to systems development. The fundamental assumption seems to be that systems development can be carried out in a standardized manner; independent of the qualifications and characteristics of the involved participants, and independent of the situation in which a given project finds itself (Kraft 77).

4.2. Limits to Formalization

Examples 4 to 7 are illustrations of the use of rules and procedures for management purposes. What are the limits to formalization?

To answer this we will have to look at the general *mechanism* which makes formalization work as a means for regulating behaviour: In a given situation there will be a rule or a procedure according to which an activity must be carried out. The idea is that the rules or procedures ensure the necessary coordination between various activities. At the same time they replace the involved peoples' reflections and save time, or they replace the involved peoples' lack of reflection and improve the quality. To make this mechanism work, a number of conditions must be met:

1. The system development process must be well understood so that typical situations can be identified and related to available rules and procedures.

2. Rules and procedures must be applicable in practice. Generally this requires that the course of a project is highly predictable.

3. Rules and procedures must be thoroughly tested to ensure quality.

4. Rules and procedures must be adapted frequently in accordance with changing environments.

5. System developers must be well trained in using the available rules and procedures.

Let us relate the examples to these conditions. In example 4 the conditions are met, but the rule is not implemented. In example 5 condition 1 is not met. The reason for the unsuccessful course of events is that management fails to understand the situation. In example 6 the problems relate to condition 3 and 5: Management does not take the procedure seriously; instead of undertaking an actual evaluation of the project groups' work, management merely accepts it automatically. The project group, on the other hand, takes the procedure seriously and uses it to evade responsibility. In example 7 the method developers never succeed in training the system developers in new methods; condition 5 is not met.

In relation to these conditions the state-of-the-art and the nature of the system development activities may be seen as follows:

1. The system development process is only partly understood. Situations can only be described reasonably unambiguously on a general level. Moreover, system development projects are carried out in environments which often are characterized by bounded rationality, ambiguities, and conflicts (Mathiassen 81).

2. The individual activities can only be described on a general level. System developers deal with analysis as well as with design; and they deal with problem setting as well as with problem solving. System developers do not deal with routine production.

3. Most methods are not thoroughly tested before they are put into use. How many independent test reports for systems development methods are available?

4. In most organizations very little effort is spent on adjusting and changing methods. Moreover, most methods claim general applicability. To be adaptable to changing environments the conditions for applying a method must be made explicit. This is not the case today.

5. Training is typically ignored. Reading the rules and procedures is in most cases assumed to be enough.

Like in the previous section it could be useful to distinguish between various types of management situations:

- *routine* situations, where the situation to be managed is understood and the type of strategy is known,

- *problem solving* situations, where the situation to be managed is understood, but where the strategy to be applied is unknown,

- *problem setting* situations, where the situation to be managed is unclear, and the strategy is unknown.

Again the basic claim is that the more we move away from routine situations, the less applicable are the tools for formalizing descriptions, and the more profound is the need for alternative approaches.

4.3. Alternatives

The alternative to formalizations for management purposes is better insight and understanding. The alternative to rules and procedures is concepts. The alternative to telling people *what* they should do, is to make them understand *why* they should do it.

In practice system developers often find themselves in unknown and unpredictable situations. There is only one way out of it: *they must plan and control their own project*. The point is fundamentally the same as in the relation between formalizations and descriptions. Basically system developers have to rely on informal and situation-determined behaviour (Davis 82). Formalized behaviour should be seen as a possible supplement, which in certain situations may improve quality and efficiency. However, it is just as important to establish a disciplined regulation of the informal behaviour.

To achieve this it is necessary, during the course of a project, to be able to describe situations a project might find itself in and compare it with other situations (Lanzara and Mathiassen 84). Furthermore it is necessary to know the causal relations between the characteristics of a situation and the conditions that created it (Munk-Madsen 84). And it is especially necessary to know possible ways of acting in different types of situations - including to know in which situations formalizations may be advantageously applied.

Finally there is the question of how working practices are actually changed. Some organizations completely ignore this problem. But the typical ways today are courses and manuals presenting guidelines without giving the participants the opportunity to try them out. An alternative would be to go through illustrative cases and examples instead of just presenting guidelines, and to establish practical experiments with new working practices in selected projects.

5. Summary

Part of literature on program development is based on a number of assumptions according to which a programming process has the following characteristics:

1. It is based on a well-defined problem.

2. Exactly one programmer is involved.

3. The result is a running program.

4. The program is to be used by the programmer himself.

These assumptions are practically never valid in system development projects - and it is probably only a minority of practical programming processes which fit into this picture. Typical system development processes have the following characteristics:

1. They are based on complex situations which may be characterized by ambiguity as well as conflict. At the same time problems continually shift during the course of the process.

2. There are several people with different qualifications and experience involved.

3. The result is new working practices and new forms of organization which entail new computer based systems and tools. A major problem is to design the interplay between these elements.

4. The so-called users rarely participate in the project, and different groups of users have different expectations and requirements to the results of the process.

This article has discussed the application of formalization in relation to the latter type of situation. One objective of the article has been to extend the discussion of formalizations from a narrow programming context to a broader systems development context. This part of the discussion has confirmed, and on some points strengthened, Naur's fundamental statement: Formalizations are of great practical value as a possible extension to a basically informal means of expression (Naur 82). Today practical systems development is based on informal means of expression, and the quality of the specifications need to be improved. This can be achieved through a more disciplined application of informal means of expression, and through an increased partial application of formalizations.

The other objective of this article has been to extend and clarify our concept of method and formalization. We see methods as prescriptions for regulating both the means of expression employed in descriptions and design, as well as the individual and collective forms of behaviour in practical systems development. There is almost always a considerable difference between

what people say they do, and what they actually do. It is our ambition to understand and change what we actually do when we program or develop systems. From this perspective it is important to see formalizations not only in relation to means of expression, but also in relation to types of action.

Concerning the application of formalizations for managing systems development, our summarizing statement is fundamentally the same as in the relation between formalizations and means of expression: System developers basically have to rely on informal and situation-determined behaviour, because they often find themselves in unknown and unpredictable situations. Formalized behaviour is just a feasible way of improving efficiency and quality in certain situations. The important thing is to establish a disciplined regulation of the basically informal behaviour.

In practice we can nevertheless observe endeavours toward formalization of systems development both on a theoretical, and on a practical level. Why? Is our analysis wrong? From our viewpoint the answer is simple: In practice programming and systems development are carried out within financial and organizational settings, and formalizations are not only applicable as professional tools for executing the working process in a more efficient and qualified manner. They are also often efficiently employed by external managers to exercise control over the working process.

> "To make the production of programs independent of individual programmers - in much the same way as cars are produced independently of individual automobile workers - various schemes have been proposed from time to time to standardize what programmers do ... structured programming offered an entirely new way of writing programs ... if managers could not yet have machines which wrote programs, at least they could have programmers who worked like machines." (Kraft 77).

There are, however, limits to regulating organizational behaviour by rules and procedures - even though this may run counter to the beliefs of the true bureaucrat. In many situations - especially when programming or developing systems - people have to go by intuition, bypass rules, or find new ways to solve the problems facing them.

References

Boehm, B.W. (1981). Software Engeneering Economics. Prentice-Hall.

Brooks, F.P. jr. (1982). The Mythical Man-Month. Addison-Wesley.

Davis, G.B. (1982). Strategies for Information Requirements Determination. IBM Systems Journal, 21. pp. 4-30.

Floyd, C. (1984). A Systematic Look at Prototyping. In Budde, R. et al. (eds.) Approaches to Prototyping. Springer-Verlag.

Freedman, D.P.; Weinberg, G.M. (1982). Handbook of Walkthroughs, Inspections, and Technical Reviews. Little, Brown and Co.

Jackson, M.A. (1975). Principles of Program Design. Academic Press.

Jackson, M.A. (1983). System Development. Prentice-Hall.

Kraft, P. (1977). Programmers and Managers. Springer-Verlag.

Lanzara, G.F. (1983). The Design Process: Metaphores and Games. In Briefs, U. et al. (eds.) Systems Design For, With and By the Users. North Holland.

Lanzara, G.F.; Mathiassen, L. (1984). Mapping Situations within a System Development Project. University of Aarhus, DAIMI PB-179.

March, J.G., Simon, H.A. (1958). Organizations. Wiley.

Mathiassen, L. (1981). Systemudvikling og systemudviklingsmetode. (Systems Development and Systems Development Method). University of Aarhus, DAIMI PB-136.

Munk-Madsen, A. (1984). Practical Problems of System Development Projects. In Sääksjärvi, M. (ed.) Report of the Seventh Scandinavian Research Seminar on Systemeering. Helsinki School of Economics.

Naur, P. (1982). Formalizations in Program Development. BIT, 22, pp. 437-453.

Polya, G. (1957). How to Solve It. Doubleday and Company.

Weinberg, G.M. (1982). Overstructured Management of Software Engineering. Course Notes for Problem Solving Leadership Workshop.