5

Formalization in Practice*

Lars Mathiassen
Andreas Munk-Madsen

Abstract. Formalizations are used in systems development to support the description of artifacts and to shape and regulate developer behavior. The limits to applying formalizations in these two ways are discussed based on examples from systems development practice. It is argued that formalizations, for example in the form of methods, are valuable in some situations, but inappropriate in others. The alternative to uncritically using formalizations is that systems developers reflect on the situations in which they find themselves and manage based on a combination of formal and informal approaches.

1. Introduction

For several years computer scientists have engaged 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 such as 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 subactivity in the wide spectrum of activities with which we are concerned. These activities are analysis, design, coding, testing, implementation, and not to forget, the important activities of systems development management.

We experience a wide gap between the discussions in the scientific literature and systems development practice. In spite of the in-

* Published as: Formalizations in Systems Development. L. Mathiassen & A. Munk-Madsen. In: Behaviour and Information Technology, Vol. 5, No. 2, 1986.

tensity of the discussions formalizations are used only 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 drawing on experiences from systems development practice (MARS Project 1984a, 1984b).

2. Methods for formalization

In our terminology a *method* consists of prescriptions for performing a certain type of working process (Mathiassen 1981). 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 1983) 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 facts that resources are limited and 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 *behavior* (when carrying out systems development and when programming). According to the *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 is merely an extension of the informal one, not a

replacement of it (Naur 1982). 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 work which implies a certain degree of formalized behavior (see figure 1).

Many discussions 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 issues related to system analysis and design. Section 4 will discuss prescriptions for formalized behavior, especially principles for organizing development activities. Here we address issues related to management.

3. Analysis and design

Descriptions—of any degree of formalization—play an important

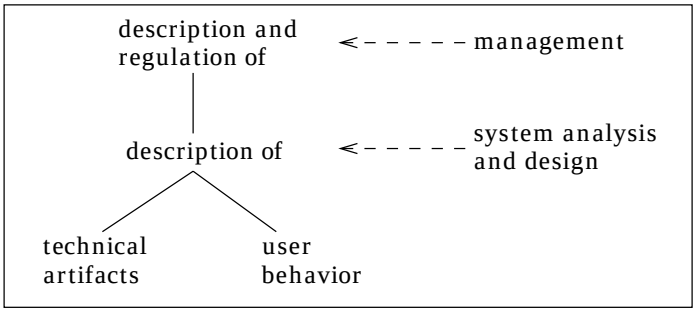

Figure 1. Formalization relates both to the level of system analysis and design, and to the level of systems development management.

role 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 developing the system: analysis, design, coding, test, and implementation. 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 implementation plan, and organizational implementation plan.

3.1. Possibilities and problems

> Example 1
> A systems 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 error 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 1982). 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 systems development effort may in fact employ any number of different formal notations side by side without contradictions. Using simple formalizations can be both 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 can only interpret and

execute formalized descriptions.

2. A formalized description can be an effective means to avoid ambiguity and obtain conciseness in communication between people.
3. A required use of formalizations can support the systems developer's understanding because they force him to think.
4. In systems development, several types of description 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 systems development situations. We find that practitioners are too poorly acquainted with the various tools for formalizing descriptions. Michael Jackson's structure diagrams (Jackson 1975) 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 criticism 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 throughout 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 systems 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 the assignment then 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 analyze

and design how reasonable estimates of throughput time could be determined with the aid of computer-based tools. The alternative would be to start analyzing the production planners' work, and how the throughput time occurred in their work.

Part of the 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 1957). 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, that is, 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 systems 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 systems developers were faced with inconsistent requirements: their assignment was to develop a system which was common for all the departments, but each department wished to maintain their individuality. A naive suggestion would be that a formalized description of the requirements would have brought this problem to the surface 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 much effort was 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.

No description tool would 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 systems developers are left with going back home to specify a more refined solution. In principle the situation could be handled by making two systems in one. This solution would, however, be neither 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 systems developers should force the organization to adopt a stance.

3.2. Limits to formalization

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, that is, the tool should capture the characteristics of what is to be described.
5. The systems 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 systems developers found a phenomenon, i.e. computation of throughput time, which was suitable for formalized description. But the application of a specific tool misled the systems developers to believe that the throughput time could be calculated. In example 3 both conditions 3 and 4 were met and the systems developers attempted to design several solutions. But they faced conflicting requirements, so what appeared to be a description problem was in reality an organizational problem.

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 at 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. Systems developers know in general too little about tools for formalizing descriptions.

Limits to formalization may also be discussed by distinguishing between various types of analysis and design situations. Inspired by Lanzara (1983) 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 systems developers.

In these terms, examples 1, 2, and 3 refer to routine, problem-solving, and problem-setting situations respectively. 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 limits to formalizations, seen in relation to the nature of the systems development process, indicate that the systems developers should have the possibility of combining different approaches in any given situation.

Formalizations are of great practical value as extensions of a basically informal means of expression. Practical systems development is very much based on informal means of expression. The quality of these specifications might improve through a more disciplined application of informal means of expression, and through an increased partial application of formalizations (Naur 1982).

There is a tendency to neglect the activities in systems development for which formalizations are unsuitable, especially the analysis of the users' work and organization. Tools and techniques for analyzing and designing the users' work and organization ought to play a more dominant role in practice and in research and more emphasis should be put on experimental strategies (Floyd 1984). Experiments can, in general, be advantageously employed to clarify analysis and design situations characterized by ambiguity and uncertainty (Davis 1982).

4. Management

We now turn to the second use of formalizations in systems development practice. This section will discuss formalizations—especially principles of organization—in connection with process design and management. We focus on those activities in a systems development project which are necessary because the project is carried out by more than one person, and which do not directly contribute to the development of the system. These activities include planning and evaluation of resources, activities, and products; regulation of conditions; configuration management; and general management activities like information and team building. Depending on the organization, some of these activities are performed by the systems developers themselves, others are performed by managers.

The central question is: in which situations does systems 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 systems developers' rather extensive experience with the application and the development environment. However, system test and conversion were estimated to only 6 per cent of the total development time. It actually took 20 per cent, which is close to the text book recommendations (Boehm 1981). 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" should have been implemented, and the advantages of doing so are self-evident.

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

1. Formalization is a means for increasing the quality and efficiency of the process.
2. Formalization is a means for increasing the efficiency of external control of the process through reports and directives.

In example 4 the first of these points is met.

> Example 5
> A project was heavily staffed-up 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 *et al.* 1982), might have given management a warning not to staff-up. There are, however, a number of uncertainties related to the proposed procedure: the review might offer an incorrect assessment; it might be decided that there is no time for a review because the project is behind schedule; and 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 1982).

> Example 6
> One organization required that a steering committee accepted the overall design. The steering committee, however, accepted an overall design which had many defects. Confronted with criticism 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 on 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 responsibili-

ties formally. But rules and procedures may also work as pretexts for doing nothing, and they can support opportunistic adjustment of behavior. They may thus—directly contrary to the intention—counteract genuine problem solving. This phenomenon is generally known as the dysfunctional effect of bureaucracies (March *et al.* 1958).

> Example 7
> The method section of a data service organization was responsible for improving practices in systems development. The section saw its main task in producing guidelines adapted to the organization. Its activities mainly consisted of looking for solutions in the available literature, in participating in courses and meetings, and in writing guidelines. The resulting guidelines were, however, 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 to ensure that the guidelines are observed in practice. In relation to the chosen strategy for improving practices, the section solves only one part of the task, while the problems related to bringing new methods into practical use are left to random initiatives.

Why does the method section operate in this way? A simple answer would be that the section is staffed with systems developers, and that they think that programmers are programmable (Weinberg 1982). There is, however, an underlying problem connected with the chosen strategy. Changing working practices is specialized as an independent function, and no attention is paid to the experienced problems related to systems development practice. 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 1977).

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?

Rules and procedures are meant to ensure the necessary coordination between individuals and activities. At the same time they can serve as a substitute for the involved actors' reflections and save time, or they can replace their lack of reflection and improve

quality. But to make this mechanism work, a number of conditions must be met:

1. The systems 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. Systems developers must be well trained in using the available rules and procedures.

Let us relate the examples to these conditions. In example 4, they - 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 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 systems developers in new methods; condition 5 is not met.

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

1. The systems development process is only partly understood. Available descriptions are typically expressed in general terms, and most projects are carried out in environments which are characterized by bounded rationality, ambiguities, and conflicts (Mathiassen l981).
2. Systems developers deal with analysis as well as with design; and they deal with problem setting as well as with problem solving. Systems developers do not only deal with routine tasks.
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 seldom the case.
5. Training is typically ignored. Reading the rules and procedures is in most cases assumed to be enough.

As in the previous section, we can express the limitations of formalizing the development process by distinguishing between various types of management situations:

- *routine situations,* where the situation to be managed is understood and an appropriate strategy is known,
- *problem-solving* situations, where the situation to be managed is understood, but the strategy is unknown,
- *problem-setting* situations, where the situation to be managed is unclear and the strategy is unknown.

The more we move away from routine situations, the less applicable are the tools for formalizing the development process, and the more profound is the need for alternative approaches to management and coordination.

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 systems developers often find themselves in unknown and unpredictable situations. There is only one way out of it: they must design and manage their own project. Systems developers have to rely on informal and situation-determined behavior (Davis 1982), and formalized behavior should be seen as a possible supplement, which in certain situations may improve quality and efficiency. It is, however, just as important to establish a disciplined regulation of the informal behavior.

To achieve this it is necessary, during the course of a project, to learn from the situations a project finds itself in and compare them with other situations (Lanzara *et al.* 1985); it is necessary to diag-

nose the causal relations between the characteristics of a situation and the conditions that created it (Munk-Madsen 1984); and it is especially necessary to know which approaches are appropriate in different types of situations—including knowing in which situations formalizations may be applied with advantage.

Finally, there is the question of how working practices are improved. Some organizations completely ignore this problem, while the typical strategy today is based on courses and manuals with guidelines. 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 the literature on program development is based on 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 systems development projects—and it is probably only a minority of practical programming processes which fit into this picture. Typical systems 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, interests, and experiences 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 this latter type of situation extending the discussion of formalizations from a narrow programming context to a broader systems development context. 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 I982). Today practical systems development is to a large extent based on informal means of expression, and the quality of the specifications needs 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 article has also extended and clarified the concepts of method and formalization. We see methods as prescriptions for regulating both the means of expression employed in analysis and design and the individual and collective forms of behavior in systems development practice. There is often a considerable difference between what people say they do, and what they actually do. If we want to change what we actually do when we program or develop systems, it is important to understand formalizations not only in relation to means of expression, but also in relation to types of action.

Systems developers basically have to rely on informal and situation-determined behavior, because they often find themselves in new and uncertain situations. Formalized behavior is just a feasible way of improving efficiency and quality in well-known and structured situations. The fundamental challenge is therefore to establish a disciplined regulation of the basically informal behavior.

We can nevertheless observe endeavors toward formalization of systems development both on a theoretical, and on a practical level. We suggest that this phenomenon is one of many expressions of the contradictory nature of systems development practice. Programming and systems development are carried out within financial and organizational settings where formalizations are applied not only as professional tools for doing the job in a more efficient and qualified manner. Formalizations are also used to exercise external 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 1977).*

But there are, as we have seen, limits to regulating systems development practice by rules and procedures—even though this may run counter to the beliefs of the true bureaucrat. In many situations, systems developers have to go by intuition, bypass rules, or find new ways of solving the problems facing them.

References

Boehm, B. W. (1981): *Software Engineering Economics.* Englewood Cliffs, New Jersey: Prentice-Hall.

Brooks, F. P., Jr. (1982): *The Mythical Man-Month.* Reading, Massachusetts: Addison-Wesley.

Davis, G. B. (1982): Strategies for Information Requirements Determination. *IBM Systems Journal.* No. 21 (4–30).

Floyd, C. (1984): A Systematic look at Prototyping. In R. Budde *et al.* (Eds.): *Approaches to Prototyping.* Berlin: Springer-Verlag.

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

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

Jackson, M. A. (1983): *Systems Development.* Englewood Cliffs, New Jersey: Prentice-Hall.

Kraft, P. (1977): *Programmers and Managers.* New York: Springer-Verlag.

Lanzara, G. F. (1983): The Design Process: Frames, Metaphors and Games. In U. Briefs *et al.* (Eds.): *Systems Design For, With and By the Users.* Amsterdam: North Holland.

Lanzara, G. F. & L. Mathiassen (1985): *Mapping Situations within a Systems Development Project.* Information and Management, Vol. 8, No. 1.

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

Mathiassen, L. (1981): *Systems Development and Systems Development Method.* University of Aarhus, DAIMI PB-136. (In Danish)

MARS Project (1984a): *MARS. A Research Project on Methods for Systems Development.* Report No. 1. Aarhus University.

MARS Project (1984b): *Systems Development in Practice.* Reports No. 2-5. Aarhus University. (In Danish)

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

Naur, P. (1982): Formalizations in Program Development. *BIT*. No. 22 (437–453).

Polya, G. (1957): *How to Solve It.* Garden City, New York: Doubleday.

Weinberg, G. M. (1982): Overstructured Management of Software Engineering. *Proceedings of the Sixth International Conference on Software Engineering*. Tokyo, Japan.