4.4 Artifacts in Software Design

Reinhard Keil-Slawik

4.4.1 Introduction

"A scientific discipline emerges with the – usually rather slow! – discovery of which aspects can be meaningfully 'studied in isolation for the sake of their own consistency'."[1] This statement made by E.W. Dijkstra was meant to express a specific desire, namely, to achieve basic improvements in software development by means of mathematical tools and concepts allowing us to express algorithms and data structures in an increasingly precise, unambiguous, consistent and complete manner. The question is, however, whether isolated mathematical properties provide the only – and a sufficient – basis for establishing a scientific discipline.

More than twenty years after the term *software engineering* was coined, the aim of turning the development of software into an engineering discipline based on sound scientific principles has only been partially achieved. Despite some progress in the development of more powerful tools and mathematically based specification techniques, the results have often been less promising than expected. Still, the quality of software is only revealed to its full extent once it is in use. Software projects fail to live up to the expectations of developers and managers or the domain experts who ultimately have to use the product. Frequently, up to three or four versions of a software system have to be delivered before it is considered reliable and sound enough to support performance of the tasks in question.

In order to understand and deal with the problems involved here, we cannot view software and its components merely as isolated mathematical objects. Behind such a strict engineering perspective lies the implicit assumption that thinking is a more or less rule-based process performed by our brain on some internally stored representations that embody our knowledge of the outside world. Once we are able to express this knowledge symbolically in the form of documents or machine-executable programs, these artifacts are said to represent or process (create, delete, modify, etc.) it. Thus, a 'transfer' of knowledge can be accomplished by exchanging artifacts, and, by the same token, human information processing can be replaced by machine operations.

However, this view does not reflect the idiosyncrasies of real software development processes. This, as C. Floyd has already pointed out in her introduction[2], involves going beyond what she has termed the traditional scientific paradigm of computer science. And this I shall attempt to do, by reflecting on the role

[1] [Dijkstra, 1982, p. 60]

[2] See Chap. 1.1.

of artifacts in design processes, in particular how they serve to support communication and learning. In order to do so, I shall have to touch on some basic philosophical questions concerning how humans acquire knowledge and how they construct and communicate meaning(s).

I argue that *thinking does not take place inside our heads but is an activity that we perform with our heads.* Most of our mental activities need external resources, and very often thinking is merely a grouping or regrouping of objects in our environment. This perspective emphasizes that humans basically use artifacts to acquire knowledge and create meaning rather than to represent it. Knowledge and meaning are attributes of cooperative social processes; they can neither be located in an artifact, nor are they stored in the brain. A document or piece of software can only be said to represent knowledge to the extent that a common framework for interpretation has been established among the parties involved. I present some guidelines for the design of artifacts that are meant to support the establishment of such a framework rather than to represent knowledge.

4.4.2 Engineering software

The technological achievements of our western civilization are chiefly built on the ability to store, modify and retrieve symbolic representations. Without the invention of mathematical formulas, specification standards or technical drawings, engineering disciplines would be practically non-existent. With the invention of symbolic representations, artifacts can be designed that would be too large to be made by a single craftsperson[3]. An important part of any engineering discipline is the development of tools and techniques and the definition of standards allowing us to create suitable design representations. To distinguish the models or representations produced while employing these means from the artifact being designed, I will call the former *design artifacts* and the latter *products.* With respect to the actual design process, the design artifacts can be said to embody the knowledge about the product being designed.

The material of which the design artifacts consist is usually different from that used for the construction of the product. With the exception of physical models or prototypes, they are symbolic representations serving two purposes: They allow the designers to explore the design space and communicate the knowledge about the product that is acquired in the course of design. Since symbolic representations can normally be created and changed with less effort than is required for the construction of their physical counterparts (i.e., the products or physical models), it is often not recognized that design artifacts can only be understood to the extent that the corresponding physical changes are understood. Essentially, this is also true of software engineering.

There is, however, one essential difference: traditional engineering focusses primarily on material structures and their physical effects, whereas software engineering is mainly concerned with symbolic structures and their cognitive

[3] [Jones, 1979, p. 124]

effects. The reason for this is that there is only one sort of material: the design artifacts and the product itself are both symbolic representations. Furthermore, programming languages are flexible and powerful means that provide an infinite variety of ways to embody system functions. Hence, the problem was to develop professional standards governing how certain phenomena should be expressed so as to enable them to be generally understood and communicated. Consequently, one of the main concerns was to get rid of the designers' or programmers' individuality and make programs and documents more accessible to other members of the project team.

Phase models and abstract machines

As a matter of fact, ever since *goto's were considered harmful*, the overriding concern has been to turn the *art of programming* into a manageable activity that uses powerful tools and formal techniques and is performed by increasing the division of labour, achieved by assigning specific functional roles to the members of a project team. This means that the knowledge embodied in a program, a program component or document must be accessible by looking at the design artifact or product in question without having to refer to the programmer who wrote it. Only then can the knowledge embodied in a design artifact or the final product be 'transferred' by handing over the relevant document.

In software engineering, the so-called phase model provides the means for combining this view of 'knowledge transfer' with the scientific ideal put forward by Dijkstra. The aim was to dissect the problem domain into isolated chunks with a view to managing software development projects as well as developing research strategies for this emerging discipline[4].

F. Selig first used a phase model to define the specific problem domains with which software engineering is concerned, namely, analysis, design, implementation, installation and maintenance[5]. B. Boehm subsequently introduced the phase model as a project management tool, later advocating its use as the first of *seven basic principles of software engineering*[6]. Using the phase model as a management principle involves three activities, according to Boehm, namely, devising and maintaining a phase plan for the project; combining this plan with a sequential development approach; and finally, using the plan to control the development. This is basically achieved by associating a document (for instance, a program or a specification) with each phase, its completion serving as a milestone in the development process[7]. To allow systematic treatment and separation of the distinct phases, specific tools and techniques had to be developed. Phases became independent domains of scientific enquiry.

[4] Software engineering as a discipline matured roughly along the lines of the phase model. Cf. [Freeman, 1979, p. 44].

[5] Cf. [Naur and Randell, 1969, p. 21].

[6] Cf. [Boehm, 1976, p. 1227] and [Boehm, 1977].

[7] The reader should bear in mind that a variety of different phase models can be found in the literature. Since I am here more concerned with the general idea than a specific instance or refinement, I will continue to use the term *the* phase model.

With respect to the design process, the milestones or documents of the phase model are the design artifacts, the installed software representing the product[8]. Since software can be regarded as a mathematical object, the idea is, then, to develop mathematical tools and techniques that allow the designers to specify the behaviour of software in a precise, complete and unambiguous manner. Once such a specification has been written, it is possible to verify whether the implementation meets the specification. Consequently, a specification of this sort can be regarded as an *abstract machine*, since it already determines the input/output relation of the software under development. This notion was originally introduced by Dijkstra as a way of devising a hierarchical software structure by designing complex general operations which are successively transformed into a combination of simpler and more specific operations[9].

The desire to arrive at a hierarchical structure by designing layers of abstract machines implies developing these abstract machines in a specific sequence of steps, because a more abstract machine defines the constraints for realization of the next-lower (abstract) machine(s). Each such step can be interpreted as a transformation from a behaviour specification (i.e., *what* should be achieved) to an implementation (i.e., *how* it is achieved). This *top-down* approach has been advocated with a view to creating design artifacts or programs (functional decomposition) as well as creating a sequence of design artifacts (phase model). According to the latter, an initial set of requirements that defines the problem space is transformed and refined into successive documents until, finally, a system is implemented, tested and installed.

However, a closer look at the idiosyncrasies of software development[10] reveals that the design artifacts cannot represent the knowledge about the product in the way suggested by the traditional engineering perspective.

Top-down considered impossible

According to our modern scientific ideal, knowledge about natural phenomena and physical structures is largely independent of its creating act, i.e., the creators and the specific setting of its creation. The experimental philosophy is a means of ensuring that the observations made and insights gained are independent of the observer. Thus, as long as the phenomena being studied are stable (repetitive) and all those involved adhere to a common framework of interpretation – such as is established, for instance, by education and training – this ideal can, to a

[8] In a strict sense, the program code would be a design artifact, and the indispensable user manual(s) would be neither nor. To avoid confusion, I will in most cases refer to both of them explicitly, using the term *document* to denote any of these artifacts.

[9] See [Dijkstra, 1968] and [Dijkstra, 1969, pp. 181–182]. Note that Dijkstra did not combine this document structure with a temporal development structure, i.e., a sequence of transformational steps. In his example of the T.H.E. Operating System, the hierarchical structure was achieved by restructuring the already finished program code.

[10] See also the detailed account given in [Budde and Züllighoven, 1990] and their summary in Chap. 6.2.

considerable degree, be maintained. The same holds for the use of design artifacts in traditional engineering disciplines. But, as I will go on to show, it does not hold for the development of software.

Traditional engineering focusses primarily on material structures and their physical effects, whereas in software engineering we are mainly concerned with symbolic structures and their cognitive effects. There are two main reasons for this difference, which are closely related to each other:

- the highly dynamic nature of the relationship between the form or artifact and the usage context, and
- the high degree of uniqueness on various levels of development and use.

As D. Parnas has pointed out, software in general lacks the degree of repetitiveness which is so characteristic of materials or artifacts in other engineering disciplines[11]. This is due in part to the complexity and dynamic nature of the context.

Traditionally, engineering problems consist in finding a new technical solution for a given function. The functionality of the automobile, for instance, has remained almost the same for more than a hundred years, but the technical implementation has improved tremendously. In contrast, a critical step in the development of software is defining and agreeing upon the required functionality of the future system. In most cases, there are different parties and user groups involved – with different roles and perspectives, and with conflicting interests[12]. Consequently, the specification of requirements may be the result of a complex process of bargaining, negotiation and evaluation. The requirements emerge as a trade-off between various interests and alternatives rather than as a self-contained specification of a technical solution to a well-known problem.

First, initial proposals are prepared and rejected. Then specifications are written and revised. Finally, programs are implemented, tested, corrected and partly restructured before the first version of the envisioned product is released. By the time the system is installed, people's behaviour and their requirements may have changed or will change once the system is in use and its quality is experienced by its users. In general, experience gained in using the system results in new insights and demands. This, M. M. Lehman argues, gives rise to a constant pressure for correction and improvement, and he concludes, "the need for continuing change is intrinsic to the nature of computer usage"[13].

If we regard software as a mathematical object that is interpreted by a machine, its semantics are a static attribute of the program text. Once the instruction sequence is fixed, the behaviour of a program is determined solely by the input. The crucial point for the developers as well as for the users, however, is determining whether a given instruction sequence is appropriate for supporting execution of the task in hand, i.e., finding out which input sequence will produce the desired output in a suitable and comprehensible manner. In addition

[11] [Parnas, 1985]
[12] See the personal account given by K. Nygaard in Chap. 2.4.
[13] [Lehman, 1980, p. 1061]

to understanding what the system should do or actually does, it is indispensable to understand what it should not do or what it does not do. Since software embodies a variety of claims and assumptions about the context and the nature of the problems to be solved by introduction of the system at the workplace[14], these properties describing the relations between software and the usage context cannot be expressed in terms of formalisms. Too many mutually influential factors have to be taken into account. The nature of the problem as it is perceived by the designers changes with every new insight, and very often incompatible requirements lead to *design conflicts* that have to be resolved.

The knowledge required for design, then, has to be built up in the course of a tedious and often painful learning process in which the designers learn which aspects fit into their already developed framework and which ones require redesign, correction or restructuring of already existing design artifacts and programs. The reasons and motivations behind such changes, and the arguments concerning how these changes are achieved while maintaining the overall quality of the design, are not part of a program or its specification, and they cannot be documented in their entirety. Programming, P. Naur argues, should not be regarded basically as an activity concerned with producing program text and its associated documentation, but as a human endeavour in which the programmers build up a theory of how the problems in hand can be solved by program execution. Naur concludes that "...reestablishing the theory of a program merely from the documentation is strictly impossible"[15]. Therefore, he argues, the meaning of a program can only be revived as long as there is at least one member of the original development team available. Merely handing over documents does not transfer the knowledge. However, if a document fails to adequately represent the knowledge required to construct the product, then neither a top-down nor a bottom-up approach will be appropriate for design.

We thus face a dilemma. Design artifacts play an essential role in every engineering discipline and therefore in any design process. In software engineering, though, they cannot play the same role as in traditional engineering disciplines. Hence, besides recognizing the problems on a phenomenological level, we must find a way to resolve this dilemma by going beyond the traditional engineering perspective.

Limits of the traditional research strategy

The idiosyncrasies outlined above reveal that software development must be regarded basically as a cooperative learning process. According to J.C. Jones, cooperative learning should be the primary purpose of any design process[16]. But if learning and communication play an essential role, we must deal with this phenomenon in a more systematic way. Tools, techniques and guidelines which are meant to document the result of a learning process are not necessarily

[14] See J. Carroll, Chap. 4.3.
[15] [Naur, 1985b, p. 258]
[16] Cf. [Jones, 1986, pp. 120–122].

equally well-suited for supporting the learning process as such. To provide a general framework for this discussion, C. Floyd has introduced the notion of the complementarity of product- and process-oriented views, arguing that the traditional engineering perspective is basically product-oriented[17]. To illustrate the impact and limitations of an exclusively product-oriented research strategy, I will introduce the notion of *learning cycles* and adapt the waterfall model to depict the ideal of this strategy.

The waterfall or phase model shown in Fig. 4.4–1 suggests that there is a 'flow' or 'transfer' from the most abstract kind of knowledge to the increasingly specific details of everyday affairs. The actual knowledge generated within each domain is embodied in artifacts such as textbooks, tools, models or specific experimental settings. Since we are normally used to talking about knowledge only when it is explicitly given, a learning cycle can be characterized as the updating or revision of the respective artifacts. A learning cycle in software development may thus be identified with the production of a new version; in software engineering it may be the development of a new generation of tools or methods. We may also regard the notion of paradigm as denoting a specific instance or kind of learning cycle within a scientific discipline in general. Roughly speaking, a learning cycle corresponds to the restructuring of knowledge about a certain domain that is embodied in an artifact.

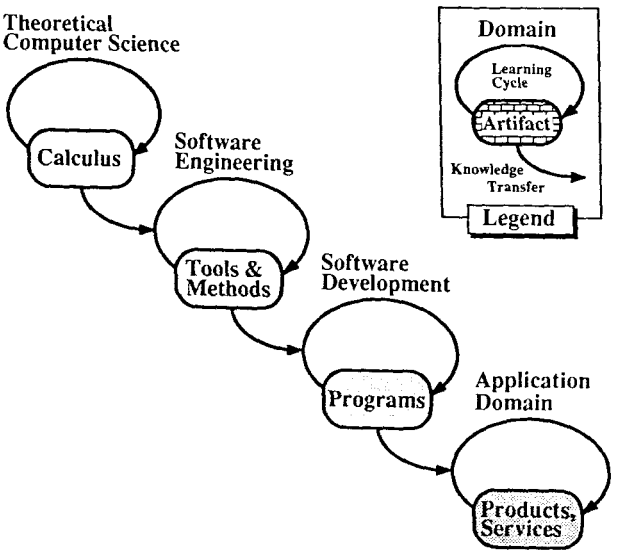

Fig. 4.4–1. The waterfall model of learning cycles and artifacts

[17] Cf. [Floyd, 1987].

In order to be able to use methods, tools or a formal specification technique, specific knowledge about the origin and inner structural relations or working principles of the utilities employed should not be required. In other words, the application of a formal specification technique should not require the competence to develop, improve, and maintain the algebraic calculus. Conversely, such competence can only be acquired with sufficient experience in the respective domain. Programmers may be capable of employing a method in which they have been trained, but they may not have the competence to develop methods on their own. And the domain experts may use word-processors to write scientific articles, but they do not have the competence to develop such systems. Hence, the development and maintenance of knowledge within a specific domain is generally associated with a specific role. The general knowledge required for applying this knowledge is acquired through training and education by those professionals who develop and maintain the respective knowledge of that domain.

The waterfall model, as outlined here, highlights the general advantage of any scientific endeavour. In software engineering, however, a crucial problem arises when this model is combined with the notion of abstract machines and a top-down development strategy. If it can be said that a design artifact is indeed a consistent, precise and complete specification of the product to be built, it has to represent all the knowledge required to construct the product. Only *implementation details*, i.e., aspects that do not alter the specification, would have to be added. Consequently, no learning is required for implementing the specification. In this case, it is, in principle, possible to execute the remaining transformational steps mechanically or automatically – i.e, to replace the human implementor by a machine.

In the course of design, where, by definition, these conditions are not given, human operations cannot be prescribed by formal procedures or replaced by machine operations. If, however, the replacement strategy is still in effect, human beings are invariably forced to perform machine-like operations that fit into the overall machine-oriented execution scheme. The typical the-machine-always-performs-better argument actually acquires validity then, because machine-like operations can be better and more reliably executed by a machine. Hence, it should come as no surprise that in software development the above guideline turns out to be counterproductive[18].

We need another perspective; the traditional product-oriented view only allows us to develop replacement strategies. To improve this situation, we have to think about how to support human learning and communication rather than replacing it. Instead of taking it for granted that a design artifact represents the design knowledge, we have to study "how to inform the material with meaning and to extract meaning from the form"[19]. The traditional perspective does not provide an adequate epistemological platform for tackling this problem, because it restricts us to viewing machines and machine-related features as the only frame

[18] Various facets of this problem are presented by D. Siefkes in Chap. 4.2 and in [Hoare, 1981, Naur, 1982, Celko et al., 1983, Floyd, 1986, Keil-Slawik, 1989].

[19] [Kay, 1984, p. 41]

of reference. In contrast, I will attempt to outline an ecological perspective by reference to our biological and cultural heritage. In particular, I will examine the role of artifacts as means for acquiring knowledge in an individual and cultural context.

4.4.3 On the evolution of meaningful forms

The notion of meaning is an inherent feature of any life form. The biologist J. von Uexküll was the first to emphasize that the recognition and creation of meaningful forms is of primary importance to every living being. Uexküll invented the concept of the *functional circle*[20] to denote that the meaning of an object is only established through the activities of a living being and has no independent existence of its own. A man, for instance, who is used to climbing up palm trees and has never seen a ladder in his life will not recognize the specific function of this device (its meaning) unless he sees someone using it or tries to use it himself. Uexküll has also pointed out that each living being is adapted with the same degree of perfection to its environment. The simple organism has a simple environment, the complex organism a complex one. Hence, the complexity and richness to which the environment may be differentiated is crucially dependent on the organisms' own inner structures.

These structures originate in an evolutionary process, which means that the more complex structure emerges from the simpler one through an adaptive process with random variations in its reproductive cycles. On the molecular genetic level, Nobel prizewinner M. Eigen and his co-workers have developed an evolution model that describes the origin of biological information as a process of selective self-organization[21]. On a broader level, cyberneticians such as H. von Foerster have developed theoretical models for self-organization and explored their epistemological consequences with respect to a broad range of scientific domains such as biology, psychology, philosophy[22].

In what follows, I will argue that the essence of perception, human learning or design is to create meaningful forms, and that this creation can be characterized as a process of selective self-organization[23].

Creating meaning

On the psychological level, this can be illustrated with reference to the notion of *gestalt*. A gestalt is often treated as a static entity or object. Its theoretical foundation, however, ties in with the notion of *self-organization*. W. Köhler writes: "wherever a process dynamically distributes and regulates itself, determined by the actual situation in a whole field, this process is said to follow principles of *gestalttheorie*."[24]

[20] [v. Uexküll, 1957]
[21] Cf. [Eigen and Schuster, 1979, Eigen et al., 1981, Eigen, 1987].
[22] See the selected articles in [v. Foerster, 1985] and Chap. 3.1.
[23] A more detailed account is given in [Keil-Slawik, 1990].
[24] [Köhler, 1935, p. 201]

A gestalt emerges when certain objects or phenomena in the environment are related to each other in a meaningful way. Unrelated physical stimuli are organized to form a coherent whole which can be distinguished from other wholes. The relation or organization as such is not present as a physical stimulus – the perceived gestalt is a construction of the observer. In general, it can be said that we perceive the world by constructing meaningful relations (gestalten). Consequently, we can only perceive what we construct.

However, these constructions are by no means arbitrary, and often not even the individual choice of the observer. The way we relate certain distinct physical stimuli to each other may be part of our subconscious body processes, i.e., fixed action schemes which we cannot influence by our will. The so-called Kanisza triangles[25] for instance, are *virtual* contours, i.e., they have the power to invoke this gestalt, and have been created to serve exactly this purpose. Why this is possible becomes apparent when we acknowledge that as human beings we have a common history and act with the same bodily means in a common environment. Hence, what is a well-adapted perceptual structure for one individual may serve the same need for any other. And what has proved to be useful in an evolutionary process may become some sort of embodied standard repertoire which does not need to be constantly learnt afresh by every individual.

Selection implies that there is a trade-off: we gain effectiveness by being able to react immediately, but pay for this with a loss of flexibility. An optical illusion, for instance, does not disappear when we know that it is one. But we can transcend this limitation. We are able to recognize an illusion through our action, by changing some part of the context and observing how these changes affect our perception of the phenomenon. We provide the required variation through our activities. As J.J. Gibson has pointed out, it is through our action that we can distinguish between what is imagined and what is real, because every close examination of real objects provides new information, reveals new features and details. A mental examination of an imagined object cannot pass this test[26].

Through our activities we are able to create ever new meaningful relations and develop cognitive structures aimed at increasing our ability to relate to the environment such that we can satisfy our needs and pursue our goals in a more flexible manner. A new cognitive structure that is formed neither by imitation nor by trial and error[27] has been called *insight* by the gestalt psychologists, and the process is called *insightful learning* [28].

Insights can be characterized as a restructuring of the perceptual field. For instance, once an ape has come to realize that boxes can be stacked on top of each other or two sticks put together to get a banana which would otherwise be out of reach, it is capable of applying this solution repeatedly, without any hesitation and to any kinds of objects which can be stacked or put together in

[25] Cf. [Rock, 1984].

[26] Cf. [Gibson, 1979]. A more elaborate discussion of human action as a validation criterion for reality is given by A. Raeithel in Chap. 8.4.

[27] It should be noted, however, that productive thinking can only take place when all these forms of learning act together.

[28] A brief description can be found in [Hilgard and Bower, 1966, pp. 229–263].

any similar situation. What has been learned by the ape is not how to stack specific boxes, but the general relation that boxes can be arranged on top of each other so as to enable it to climb up and get what it wants.

The same holds for human learning: the meaning of a form – the gestalt – is a construction of the observer. Consequently, it is not the environment that changes, but the way an individual relates the objects and phenomena in its environment to each other to form a meaningful whole – a gestalt, an organization, an architecture, or whatever.

Once an insight has emerged, we have not created yet another cognitive structure, but have revised, modified or enhanced the way we relate the things in our environment to each other. The new cognitive structure supersedes the old one[29]. Hence, gaining experience, learning to better adapt to the environment in order to achieve a goal or to satisfy a need, is not merely a matter of storing more and more cognitive structures in the same way as data is stored in a computer. And problem-solving is not a question of finding an internally stored structure that matches the problem structure. If this were the case, it would take longer and longer to search for the appropriate structure, the more experienced we were. Eventually, we would be unable to react at all; evolution would be a dead end.

Instead, the reverse is the case: the more expert we become in a particular domain, the faster we are able to identify a problem and the closer we come with our first 'guess' to the final solution. This ability is the result of an evolutionary process. Knowledge is historical in the sense that we can only make it explicit and communicate about it properly if we are able to study the learning process which established this knowledge, i.e., the way the individual being related to its environment in its complete course of events[30]. Since we are unable to make such knowledge explicit by expressing it in terms of our actual environment and how we relate the entities (which include, of course, symbolic representations) in this environment to each other, we characterize it as a different kind of knowledge. Basically, it can be characterized as the difference between *knowledge* and *competence* (or *skill*)[31]. In a sense, it can be said that intuition and feeling are our most advanced means of intelligent behaviour[32].

---

[29] This is the same characterization T.S. Kuhn has given (in the addendum of the second edition) to characterize the effect of a paradigm [Kuhn, 1970].

[30] The same holds for biological information. B.-O. Küppers points out that the information embodied in the genes cannot be derived exclusively from the genetic code; it is only given in relation to the environment. In an evolutionary process, it is selectively evaluated against the external information embodied in the environment. Cf. [Küppers, 1983].

[31] Other authors have made this distinction by contrasting different notions, such as *tacit* and *articulate knowledge* [Polanyi, 1967], *knowing that* and *knowing how* [Ryle, 1983], *symbolic reasoning* and *intuition* [Dreyfus, 1979], or by referring to the *paradigmatic* and the *narrative* modes of thought [Bruner, 1984].

[32] On the role of intuition see [Dreyfus and Dreyfus, 1986] and, with respect to software development, [Naur, 1985a].

To sum up: although we characterize an insight in terms of a specific relation of objects or phenomena in the environment, it is always the construction of an individual person. Strictly speaking, knowledge and meaning are neither qualities of the external world, nor are they stored in our brain in the same way as data is stored in a computer. Knowledge and meaning are the ways we relate to our environment. Since they are constituted as self-organizing processes, the creation of knowledge or meaning can neither be controlled nor prescribed. And there is no direct way of transporting meaning or information, giving it to another person as one hands over an artifact, a book or a technical drawing. We can only provide an environment in which the entities that have to be related to each other are present in the perceptual field or within reach.

However, merely relating things in our environment to each other in a specific way does not allow us to transcend the constraints imposed by the given environment and the restrictions of our bodily capabilities.

Artifacts as external memory

To perform so-called mental operations, we are much more dependent on our physical environment, and consequently on our bodily actions, than is generally acknowledged. Our perceptual faculties, for instance, are quite limited. By direct perception, i.e., without starting a counting or calculation process, we can only distinguish up to four items. As G. Ifrah points out, all additive numbering schemes (symbolic representations of the tally system) of different cultures introduce a new symbol by the fifth position at the latest[33]. This allows us to group the symbols on a higher level, thus enabling us to perceive greater numbers more easily under the same perceptual constraints.

Almost every calculation or counting process, however, requires the use of perceivable physical means, be they visible symbols or tangible objects. To begin with, a tally or small calculating stones were used, later on the abacus, the Indian decimal number system (IDNS), and finally algebraic formalisms, Turing machines, formal languages, etc. The word calculus, for instance, stems from the greek calculi which means *chalk pebble*. And the notion of a formal *language* is, according to S. Krämer, already misleading. What mathematicians and computer scientists develop and work with are, strictly speaking, formal *typographies*[34].

The modifications of the physical appearance (states) of artifacts – such as the positions of the pebbles on a calculating board, the marks on a tally or symbols on paper – are an indispensable part of our mental activities. The states of a tool, as well as the calculations performed with a pencil on paper, serve as an external memory which allows us to check the (interim) results, and to reflect on the process as such. Calculating reliably on an abacus, for instance, becomes increasingly cumbersome, the greater the numbers are. Without *storing* intermediate results, i.e., making them perceptible beyond the performing act, checking may become impossible because, with every calculation, the previously

[33] [Ifrah, 1987, pp. 169–183]
[34] [Krämer, 1988, pp. 176–183]

achieved result will be destroyed. The only way to check a calculation is to store the result and compare it with another calculation. This problem changes with the introduction of (formal) typographies such as the Indian decimal number system.

When we perform an arithmetic calculation with pencil and paper, we spatially arrange digits on the paper in a systematic fashion to form numbers and columns of numbers representing intermediate results. Where necessary, symbolic operators are inserted. Once we have these physical traces of the process, it is possible to discover structural relations and invariants by relating different calculations to each other. At the end of the 16^{th} century, the invention of algebra and, parallel to this, the construction of the first calculating machines served to represent the then accomplished gestalten and insights by physical means. Both, the replacement of numbers by letters as well as by gears, shafts and cogwheels, are physical embodiments of a relation which formerly had to be established by the human mind for every single calculation.

Now, it is possible to reason on the level of structural relations and make the respective consequences visible. The commutative law, for instance, represents an invariant of the calculation process and can be visualized in the written form:

$$a + b = b + a$$

This expression asserts that the equation holds for all possible instances of a and b within a given mathematical framework. By defining operations that preserve the validity of an equation, this kind of physical representation and its respective operations of generating and arranging symbols in a specific way open up a new realm of thinking. With the invention of boolean algebra, for instance, it became possible to calculate logical deductions.

Finally, a concept such as the Turing machine provides the means to represent the physical operations of transforming and arranging the symbols according to an explicitly given set of rules in the same symbolic medium. Once we are able to describe the symbol manipulation operations as a composition of elementary (atomic) symbol transformation processes, this sequence of transformations can be performed by a machine composed of elementary mechanisms resembling the atomic symbol transformations.

What is replaced by the machine, however, are not the mental activities of forming meaningful relations, i.e., the creation of gestalten, but the physical operations that modify the appearance of the respective artifacts. Once we can describe the invariants of the physical transformation processes that are part of our mental activities, we can try to devise more efficient means to express and technically accomplish the corresponding state transformations.

According to A. Leroi-Gourhan, it can be generally said that the evolution of the human mind is basically the evolution of its expressive means[35]. These expressive means or artifacts embody a new quality. As the result of insightful learning, they are more than the sum of their parts. Neither the invention of zero nor that of the bow and arrow could have been achieved by imitating

[35] Cf. [Leroi-Gourhan, 1988].

something which already existed. And there was no sequence of development steps or interim results that enabled the artifact to be deduced systematically.

Again, the notion of external memory is crucial to communication and learning. One essential difference between animals and human beings is not the construction of tools – that animals do as well – but their preservation. This is essential, because only then does it become possible to compare a previously built tool with a new one, to communicate about tools, and to use them as a means for education. All these aspects are essential prerequisites for making progress in the design of new tools. There is no straightforward way to derive a tool which will satisfy a certain need merely by individually performing internal mental activities.

So far, I have emphasized that, owing to their physical nature, artifacts function as external memory, thus facilitating communication and learning. They evolve as part of a functional circle which I have characterized as a process of selective self-organization on various levels, ranging from individual problem-solving through cooperative learning to the evolution of culture. As I have mentioned before, selection implies evaluating a trade-off. I will now discuss this trade-off function in order to identify the features of an artifact that provide the selective advantage.

Flexibility versus iconicity

Two subsections earlier, I have given an example of a trade-off function with respect to embodying human capabilities – for instance, organizing physically unrelated stimuli as action schemes. Such schemes cannot be influenced or controlled intentionally; greater effectiveness is paid for by a loss of flexibility. One could say that, in an evolutionary process, these schemes represent the memory, preserving stereotype behaviour, i.e., fixed action schemes that have proved successful in the sense of leading to a selective advantage in the past.

However, there is only a pay-off as long as the meaning of the scheme is fixed. With respect to the functional circle, this implies that the environment does not provide unanticipated events that would require changing the action sequence embodied in the scheme. If this happens, such an action scheme would lead to erroneous behaviour that might have serious consequences for the individual. But if, as I have pointed out, these schemes are embodied in a more flexible framework of learning, they will enhance the overall flexibility. When they are part of an individual's response to changing environmental conditions, they reduce the amount of cognitive effort required for controlling and performing the overall action, thus freeing the human mind to concentrate on the change pattern. In general, they decrease our dependence on environmental conditions because they give us free time which the human mind can use to develop artifacts. And this – besides offering the already mentioned advantages – may also help us to transcend the limitations and constraints imposed by the inflexibility of these schemes.

If we now view artifacts as the (external) memory of our cultural evolution, we will find that they serve the same purpose. With every new artifact – from

the tally to the abacus, the Indian decimal number system, and finally algebra and Turing machines – the sequence of bodily operations required to obtain a result has been reduced. With the tally, every act of making a mark corresponds to the act of counting an object. With the abacus, the spatial arrangement of beads allows us to move one bead into a specific position to replace the respective number of counting acts. The sequentialization of the counting process is reduced by introducing a new spatial structure. With the symbolic representation of numbers, the handling of any number of one to nine beads is reduced to the manipulation of one single symbol (digit). This also allows us to represent the concept of zero as a physical symbol like any other number. Finally, with the invention of algebra it becomes possible to embody the structural properties of an infinite number of calculations in terms of a single symbolic description. By expressing these structural properties through physical forces, it becomes possible to mechanize and later – with the invention of Turing machines and computers – to automate them, i.e., to perform a calculation by pushing a button.

As a result, more powerful operations can be performed in less time, with greater flexibility and reliability. But there is also a new quality: enforced or prescribed sequences of operations that do not allow us to create a gestalt, but which nevertheless have to be performed, are condensed into single objects or operations that can now be flexibly arranged anew and related to each other to form new gestalten or insights. In this sense, the selective advantage in the evolution of artifacts is that prescriptive temporal structures are dissolved by creating physical objects and corresponding spatial structures, or – in more abstract terms – by providing a state space that allows us to find out how we have to relate the states to each other to form new meaningful wholes. With respect to human actions, this cultural achievement can be stated as a general guideline for the design of artifacts:

MINIMIZE THE AMOUNT OF ENFORCED SEQUENTIALITY NEEDED TO ACCOMPLISH A TASK OR SET OF TASKS.

Enforced sequentiality means either that unnecessary actions have to be performed to form a gestalt or embody it with physical means, or that there is a requirement for certain actions to be performed in a given order. Either one impedes the formation of a gestalt.

Consequently, we can say that artifacts that are meant to support learning must put the user in control, i.e., enable him to plan, control and initiate the sequence of state transformations to be performed. In some cases, however, we may also learn something by following a given sequence such as is imposed by imitation[36]. The individuals still have to create the meaning, but they choose not to control the state transformation of the external memory. This may help stimulate unanticipated insights, but it is too inflexible to provide general support for the learning process.

[36] A prescription only works if the individual is willing to follow it, which again is some kind of imitation. We may put pressure on individuals to enforce such a decision, but we cannot enforce insights.

The same holds on the symbolic level: the notion corresponding to imitation is iconicity. We speak of iconicity when the pronunciation of a bird's name closely resembles the sound of its voice, as in the case of the cuckoo. Another more widespread example is the use of icons or pictograms representing an image of the object they denote. Pictograms may promote understanding by referring to an already known visual gestalt, but they do not provide sufficient flexibility for creating new meanings and embodying them in physical means. Thus, part of our cultural development has been the shift from pictorial expressions to languages based on an alphabet consisting of arbitrarily chosen symbols.

This is the price we invariably have to pay for flexibility, namely, that the artifacts we employ become less and less meaningful in the sense that the degree of iconicity is reduced. Each mark on a tally, for instance, represents a counted object. This is no longer the case when arranging beads in a two-dimensional structure by using a calculating board or an abacus. And a digit is of an arbitrary shape that in no way reflects the amount it stands for. Letters in an algebra or cogwheels in a calculating machine dissolve the notion of number and amount even further. And finally, the concept of the Turing machine reduces everything to a sequence of elementary operations by which arbitrarily chosen symbols (the alphabet) are read from and written on to an endless tape.

At every stage in this historical process, the human mind first has to find a way of arranging the visible or tangible objects to form meaningful relations in the specific context of activities. Once we are able to express these relations as perceivable objects by writing down rules and structural expressions, we can perform the physical operations for manipulation in a mechanical way. To obtain the result and to perform the operations, no conscious interpretation is needed, no gestalt has to be established or insight acquired for completion. The operations performed have become meaningless, and they only acquire meaning insofar as they are executed as part of other human activities.

This general view is in accordance with the careful distinction between *data* and *information* as defined by the IFIP[37]. Data can be transmitted and multiplied, but the process which establishes the meaning, i.e., produces the respective information, has to be carried out by every individual anew. Furthermore, data can only be interpreted by establishing conventions and standards. The social processes of defining, revising, applying, reading and teaching such standards and conventions establish a common history among the parties involved; it becomes part of their cultural environment and fosters mutual understanding.

Information, meaning, gestalten or insights are invisible by their very nature and are brought to the surface only through human activity. In order to support this activity, we have to provide artifacts which help us to make the invisible visible[38].

[37] A thoughtful account of this definition is given in [Naur, 1974, pp. 18–31]. See also the extensive characterization of information by K. Fuchs-Kittowski in Chap. 8.5.

[38] A. Kay has used the notion of visibility slightly differently, namely, to highlight the difference between the program text (visible) and what will happen during program execution (invisible). Cf. [Kay, 1984].

4.4.4 Designing software

The ecological perspective presented here emphasizes that artifacts are indispensable means for creating meaning and supporting learning and communication. Conversely, the design of a product is basically a process of cooperative learning[39]. To highlight the differences between this and the traditional engineering perspective, I will again use the notion of learning cycles to characterize the research strategy associated with this ecological perspective.

An ecological approach to software development

The basic difference between the ecological and the traditional engineering perspective is that learning cycles *within* software development and use are now acknowledged as primary means for promoting understanding and supporting communication. Its frame of reference is not based on the idea of context-free knowledge that is transferred by the exchange of (design) artifacts, but on the concept of person(s)-acting-in-settings as a specific instance of the functional circle[40]. We can then reconstruct the waterfall model into an ecological model of nested learning cycles as depicted in Fig. 4.4–2.

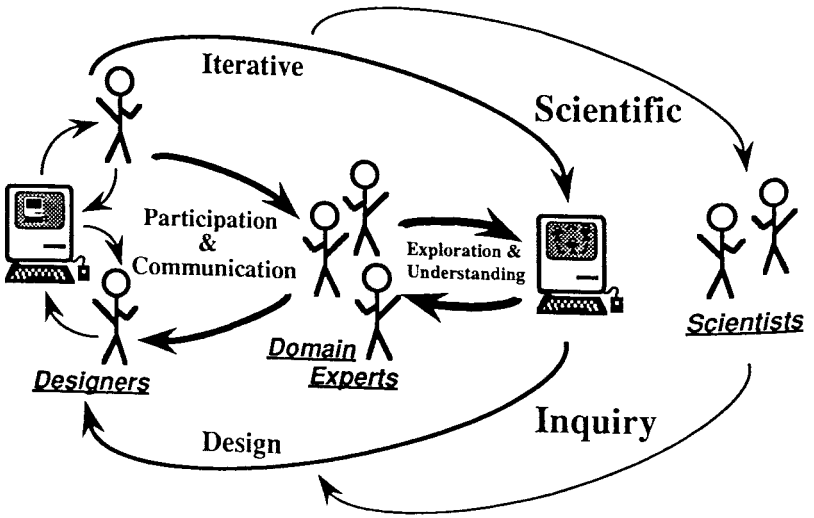

Fig. 4.4–2. The ecological model of nested learning cycles

[39] Cf. the definition of design as "the interaction between understanding and creation" in [Winograd and Flores, 1986, p. 4].

[40] This concept has been employed in particular by J. Lave to study and describe *everyday cognition*. See [Lave, 1988] and [Rogoff and Lave, 1984].

In the ecological model, the subjects of study are the processes into which the artifacts are to be embedded, and not the inherent properties of artifacts as isolated entities. To better understand what actually matters in a specific situation and to find better ways and means of supporting the design process, we have to study the human activities of developing and using software.

This is also the basic philosophy underlying the methodological framework STEPS (Software Technology for Evolutionary Participative System development)[41] developed by us at the Technical University of Berlin. In STEPS, it is acknowledged that the quality of software cannot be defined without reference to the development and usage context. This is not only essential for qualitative approaches such as case studies, but for quantitative investigations as well. Software measurements, for instance, can only be understood and interpreted with respect to a specific design setting[42].

The ecological perspective emphasizes that the result of self-organizing processes such as cognition, learning, design, or evolution can only be fully understood by reference to their history. What this means in terms of design is establishing a common history among those who are meant to understand the product. And, since learning and communication are essential for design, a participative development approach is advisable. This requires that the participants are – to a certain extent – able to pursue the matter according to their individual goals, objectives and personal needs. Thus, finding ways of *sharing responsibility*, as explored by G. Bjerknes[43], and developing a *subject-oriented approach*, as does M. Nurminen[44], are not only promising attempts at dealing methodically with the social aspects of design, but also provide ideas on how to improve design of the products.

In his book "Notes on the Synthesis of Form", C. Alexander states: "the ultimate object of design is form". In real-world situations, he points out, the problem with design is that we are trying to invent a form to fit into a context which we do not fully understand. This is especially true of the development of software. Consequently, it is not just a form but a variety of forms which are developed, revised, enhanced, or rejected in the course of software development. Basically, these may comprise the design artifacts which are produced by applying different tools and techniques, prototypes, and eventually the product and its documentation. These forms are related to each other in various ways, and changes in one form have consequences for one or several others. In addition, different people may be responsible for developing and maintaining different forms. Thus, C. Floyd's characterization of design as "a web of design decisions"[45] gives a more appropriate account of the actual process. This web, as it is physically embodied in the design artifacts and products or prototypes, normally changes very dynamically at the beginning, and becomes then more and more stable,

[41] An overview is presented in [Floyd et al., 1989b] and [Keil-Slawik, 1987a], see also the contributions of C. Floyd in Chap. 3.2 and M. Reisin in Chap. 7.3.
[42] Cf. [Basili and Perricone, 1984] and, in particular, [Basili and Rombach, 1987].
[43] See Chap. 7.1.
[44] See Chap. 7.2
[45] See Chap. 3.2.

until, at the end of the development process, the final product is released. The general guideline for the development of design artifacts and tools that are meant to support this process is to provide means to embody and maintain the web of design decisions such that the amount of enforced sequentiality is minimized.

On the basis of this view, I will now discuss how this general guideline translates into features and attributes of design artifacts and products.

Improving design

Our task is to devise design artifacts and tools so as to provide sufficient support and sufficient orientation without prescribing the course of actions to be taken. This requires means which allow us to embody gestalten in such a way that they provide a constructive basis for establishing a common understanding of the problems in hand and the desired solution.

Unlike the traditional engineering perspective, where a design artifact is supposed to be unambiguous, consistent, precise, and complete, the general guideline only demands that design artifacts – especially at the beginning of the design process – allow us to embody only those gestalten or items of information which are necessary in the specific situation to continue the (cooperative) learning process – and nothing more. Process-oriented development models[46], prototyping strategies, the development of a project language by establishing a dictionary containing the technical terms of the participants' domain languages – all of these serve this goal, as does the use of *base lines* or *reference lines* instead of phase model milestones[47]. They provide the opportunity to iterate on specifically chosen problem domains or aspects independently and on various levels of detail. In contrast, the phase model approach is transformational: each iterative step comprises the transformation of the whole problem domain.

This difference also applies to the design of products (tools) for the development of design artifacts. A tool may either only accept consistent data records as input, or it may provide a function for checking the consistency or completeness of a specified set of records whenever it seems necessary. The former allows the designer to enter only complete data records that fit into the already developed framework, whereas the latter allows him to store partial results which are not yet consistent, but may nevertheless be useful for exploring the problem.

What all these examples have in common is that they provide means to utilize the external memory to the extent required by the actual needs of the people involved without prescribing the form or structure that should be achieved or the way in which it should be achieved. In the traditional phase model approach, the latter is derived from the structure of the product.

The same idea has been expressed in a slightly different way by D.E. Knuth who has developed a tool called WEB allowing programmers to separate the final structure of the code as required by the programming language from the structure they choose during development to suit their needs and preferences.

[46] See [Floyd, 1981, Floyd and Keil, 1983].
[47] See [Andersen et al., 1990, Floyd et al., 1989b].

According to Knuth, the basic idea is to write programs not in order to instruct the machine, but rather to explain to other people what we want the machine to do for us[48]. The point is that now the grouping and sequencing of what forms meaningful wholes in the course of design is left to the designers and their understanding. Thus, the structures as required by the programming language impose less sequencing on their activities.

On a more general level, principles such as *user control*[49] or *minimalist instruction*[50] are design guidelines that serve the same end. And they can be applied to the product as well as to the design of user manuals[51].

As regards development of new products, I wish to point out that the explicit goal of providing support for individual problem-solving and information organization lead to the notion of interactive systems and, eventually, to two basic innovations: hypertext technologies and object-oriented systems. In particular the definition of hypertext as non-sequential text processing explicitly confirms the guideline for reducing enforced sequentiality. Both technologies implement the same basic idea: they allow domain experts to easily embody mentally established relations in physical terms (links, shared code) and build on these embodiments later on. However, besides assessing the essential quality of innovative technologies, the general guideline presented here can also be used to derive more specific design criteria that can be fruitfully applied in the design of use interfaces[52].

4.4.5 Summary

The ecological perspective presented here seeks to provide guidance and orientation in identifying problems and to help direct the search for solutions. It is not meant as a theoretical framework allowing us to deduce or determine the desired properties of either specific design artifacts and products or specific development methods[53]. Nevertheless it does provide some ideas on how to improve the design process.

I have characterized design as a cooperative learning process. The result or outcome of this process cannot be described precisely until the product is finished. The value of any innovation can only be defined once it has been realized and appraised, whether it be a new function or algorithm that is to be implemented, a new method to be used, or a new system to be developed. The same holds for user actions in a learning situation. In a more general sense, it can be said that the meaning of any activity cannot be described precisely until the action has been completed[54].

[48] Cf. [Knuth, 1984, Bentley, 1986]; an elaborate example is given in [Knuth, 1986], see also Chap. 1.2.
[49] See W. Dzida, Chap. 7.4.
[50] [Carroll, 1990]
[51] Cf. [Carroll et al., 1987].
[52] A more extensive discussion can be found in [Keil-Slawik, 1990, pp. 47–70].
[53] This substantiates the arguments of J. Carroll in Chapter 4.3.
[54] This is also the central theme in [Weick, 1979].

Since learning is regarded as an evolutionary process of selective self-organization, artifacts that are meant to support this process must provide means to flexibly create and embody gestalten according to the insights acquired by the parties involved. Thus, it is no longer the mathematical attributes of the product that constitute the frame of reference, but the cooperative learning processes that are part of design. Artifacts are viewed as embodying the external memory of human cognitive processes. By studying the evolution of artifacts in a cultural context I was able to derive a general guideline for their design, namely, to minimize enforced sequentiality.

As is the case with all design principles, this guideline can neither be considered in isolation, nor can it simply be optimized along a one-dimensional scale: the more flexible, the better. It is dialectical in its nature because every embodiment of a gestalt – such as the fixation of a problem, the choice of a certain function to be implemented, or a selected module structure – imposes constraints on the subsequent actions and limits the possible choices. On the other hand, without any such fixations no progress could occur. Thus, the crucial question is where and when to draw the line so as to find the right balance between flexibility and stability.

This question can be generally answered by the ethical imperative of H. von Foerster: "Act always so as to increase the number of choices."[55] And this is exactly what should be achieved by minimizing enforced sequentiality. On a practical level, however, it can only be answered with respect to a given context. In the course of developing or employing interactive systems, for instance, this is the analysis of the work environment. However, even then, as I have already pointed out, it is not possible to deduce a solution purely from the needs or requirements. We need the traditional engineering perspective as well. Without the results being produced along these lines, we would not be able to pursue our goals. Every single interactive step, for instance, embraces already a vast amount of formal operations embodying general insights that are invariant with respect to the specific setting or problem – and thus, may not have been derived from the specific context.

Both perspectives – the traditional and the ecological – must be regarded as indispensable for our scientific endeavour. Any practical design activity requires that they be productively combined. Only by their combination can we find appropriate ways of minimizing enforced sequentiality with respect to the development *and* use of software.

Acknowledgements

I would like to thank Christiane Floyd, Rodrigo Botafogo, Kim Halskov Madsen, and Ben Shneiderman for their constructive criticism throughout the various versions of this article. They have been instrumental in shaping my ideas. My thanks also go to Phil Bacon for polishing up the text idiomatically and stylistically.

[55] [v. Foerster, 1984, p. 308]