Final Manuscript for the contribution in: Y. Dittrich, C. Floyd, R. Klischewski (eds.) *Social Thinking – Software Practice*. Cambridge, MA: MIT Press, 2002, pp. 5-28.

Developing and Embedding Autooperational Form

Christiane Floyd

Relating social thinking and software practice as we set out to do in this book is a complex endeavor. What do we mean by software practice? How do we want to deal with it? What kind of social thinking do we have in mind? How do we see social thinking related to software practice? Each contribution in this book provides its own answers to these questions. The objective of this chapter is to propose a conceptual framework for the discussion of computer artifacts in connection with human activity. But first I will give some orientation on how social thinking and software practice can be related to one another.

1 Computing: Bridging the Gap between the Social, the Technical and the Formal

In what follows the term *software practice* refers to the sum total of activities involved in developing and promoting the use of software. The focus is on *software embedded in social contexts* such as human work and communication. This is not meant as a distinction between these systems and other types of software, but rather as a vantage point from which to look at software. The relevant actors are software developers, managers, vendors, buyers, and users, who, in turn, are engaged in practices of various kinds. The assumption that software practice can be meaningfully considered on its own is not at all obvious. One implication is that the hardware required belongs to a separate area of concern; another is that it is necessary to draw a fuzzy borderline between software practice and related areas such as work design and organizational development. Lastly, it should be emphasized that software practice includes all activities promoting software use (for example by coaching, involving key-users, or providing technical infrastructure), but not software use as a whole, which occurs as part of diverse human practices.

Software practice varies greatly from one setting to another. For example, we can distinguish research from industry, technology innovation from technology application, the production of new software from the adaptation and enhancement of existing software, the development of standard software from the customization for individual use sites, and so on. Questions of scale also matter: in large firms, software practitioners tend to be in highly specialized work settings, while in small firms generalist jobs are more common, or the firm as a whole is specialized. Also, software practice is shaped by how individual firms cooperate with one another on a local or global scale.

The present discussion is not primarily concerned with portraying the variety of activities involved in software practice. Instead the emphasis is on the objective of software practice: software as artifact to be designed, developed, introduced, or promoted within software practice.

Therefore, cutting across the distinctions made above, I propose to distinguish *levels of software practice* depending on the complexity of the social context relevant to software. These levels of concern, coexisting in time and often nested in one another, require different kinds

of skills and give rise to different areas of work for both computer professionals and users. They also entail quite different problems for social reflection, so I suggest viewing social thinking in connection with these levels of software practice.

Against this background I address my topic, which is a conceptual framework for understanding computer artifacts highlighting how computing is anchored and how computer artifacts are embedded in human activity. I hope to contribute to ongoing discussions on the nature of the discipline of computing–its object, its scientific core, and its methods–with a view to promoting a discourse with social theory. The sheer diversity of names (*computer science*, *computing science*, *informatics*, *datalogi*...) as well as the separation in some countries, for example between computer science and information science, point to the fact that there are various complementary or even conflicting views of this discipline. To avoid these terminological difficulties, I shall use the term *computing* throughout this chapter in a very general sense.

I will portray computing as a constructive discipline around the development and use of computer artifacts that at its core is concerned in a specific way with form. The notion of form relevant here needs to connect organized human activity and technology development with the strictly logic-based idea of formalization inherent in theoretical computer science. Bridging the gap between the social, the technical, and the formal realms with the concept of *operation*, I will introduce the notion of *operational form* as a key to understanding what can be reified in computer artifacts. On that basis, I will argue that computing is concerned with producing and embedding *autooperational form*. This means (1) making operational form explicit in the area of interest and (2) making autooperational artifacts available for human use.

In connection with this I will uncover *operational (re-) construction* as the basic method associated with computing. That is, when engaged in computing, we perceive the area of interest–any part of the natural, the social or the technical world or a formal universe of discourse–in terms of operational form. Of this we develop a decontextualized model, which, implemented as a program, becomes autooperational when executed on a computer. Thus, choices in design reflect *how* we perceive the area of interest in terms of operational form. While computer artifacts may be designed, for example, as tools, media, or agents, the distinguishing property of these tools, media, or agents is that they are autooperational. When embedding computer artifacts, we create conditions for recontextualizing autooperational form and open up spaces of opportunities for situated human activity.

The view proposed here brings out the inherent connection between design and scopes for choice. While human activity is not determined by computer artifacts, as autooperational form is always interpreted in the use situation, it unfolds in the space of opportunities open to technology users. Thus we make decisions on available options in design. This clearly relates to issues of responsibility to be negotiated as part of software development.

The double purpose I pursue in this chapter is reflected in the structure of the text. Section 2 may be considered a metatheoretical contribution: I will introduce the field of relating software thinking and social practice in a general way, indicating basic questions of interest and distinctions between different social theory approaches as they pertain to software practice. In sections 3 and 4 I present my considerations on developing and embedding autooperational form. Section 5 ties the two parts together by locating the specific conceptual framework I propose in the overall territory of social thinking and software practice sketched earlier.

Both in the metatheoretical and the conceptual framework sections, the discussion relates to the works of many authors in computing, social theory and the philosophy of science, far too many to refer to in an introductory chapter. As an alternative and not without regret, I opted for presenting the argumentation itself, and just point to connections with theoretical stands taken elsewhere. I know this to be problematic but I hope to make the text more legible this way. [1]

2 Viewing Software in social contexts

To understand how social thinking can be related to software practice, we need to look at software in social contexts. Software development and use, even in everyday settings, necessitate taking a stand on basic questions relating computers to human beings such as: Can computers, in principle, be likened to human beings? How can the social world be modeled in computing terms? How should computers be allowed to interface with human activity? These questions arise at different levels of complexity in design and the stand we take on them is embodied in design decisions. On the other hand they are dealt with by social theories in different ways. In what follows, I aim to characterize the problems at hand in terms that will hopefully enable the reader to gain a deeper understanding of his or her own questions.

2.1 What is a computer?

Let me start with a little anecdote. At a conference on social changes related to information technology the lecturer, a computer scientist, gave an excellent overview about the most recent technical developments, focusing on advanced Web applications. But the first question in the discussion was simply "What is a computer?" Raised by a white-haired gentleman who has lived through the whole era of computing, this question caused considerable embarrassment. Asked in the twenty-first century, it seemed unbelievably naive. Or was it?

After some hesitation the lecturer described the computer as a technical device that can do computations "and of course much more." Clearly he was trying to condense his technical and formal knowledge into a few simple words suitable for a non specialized audience. His aim was to describe the computer as an entity in itself, endowed with capabilities provided through programming. As seen from his angle, the basic paradigm of computation, extended from the numeric to the symbolic and the structural realms, gives rise to the whole spectrum of possible programmed behavior.

I interpreted the question instead as relating to issues such as: What parts of the real world can be modeled in computer programs? What kind of reduction is implied? How can computer programs become effective in the area of interest? How are computers perceived in interaction? These issues are fundamental to understanding computers *in use*. They need to be addressed in a way that permits to make the connection between the full world of human activity and experience and the abstract world of the computational paradigm. I also gave a simplified answer, describing the computer as a technical device capable of storing and executing predefined plans for action.

The gentleman did not speak up again and I do not know which answer came closer to what he had in mind. It struck me how difficult his question was to reply to in spite of some fifty years of debate on humans and computers. Also, his concern was still with "the computer"–

perceived as an individual technical agent exhibiting quasi-intelligent behavior–rather than with information technology as a whole.

This incident again confirmed my suspicion that all social reflection on computing rests on understanding computers in the human context. But unlike the lecturer just referred to, I am not interested in dealing with these questions in ontological terms. Instead, I would like to understand computers as artifacts and propose to bring in human beings involved in programming them, interacting with them, using them, and so on, as the frame of reference. This implies keeping track of two distinctions. One is Heidegger's distinction of experiencing something as "thing-in-use" (*Zeug*), as "ready to hand" while being immersed ("thrown") in an ongoing activity, as opposed to understanding it as "thing" (*Ding*): an entity "present at hand" with properties that can be described and studied on their own. The other is that between development and use perspectives. For software professionals the object of their work is to find and implement a computational structure meeting the needs of the users. For users the results of these efforts provide the technical means or environment for performing their own work. Thus, following Leigh Star, software can be regarded as border object shared among different communities of practice.

To appreciate how software mediates between developers and users, we need to be concerned with software development as a human activity, software as a means of reifying aspects of human activity, and software use in the context of human activity. On the theoretical level, this will necessitate an encounter between formal theory underlying the computing field, and social theory about human activity in software development and use.

2.2 Formal and social theory perspectives on computing

As we have seen, it is not easy to find appropriate general terms for what is of interest. Should the focus be on "the computer"? This is misleading, since it gives the illusion of a stand-alone agent, and obscures how the computer as an agent is instructed by a multitude of interleaved programs. "Software"? No, software is never experienced in itself. "Computer-based system"? Then there is the suggestion of a system, comprising technical elements and more–very problematic. I opt for "computer artifact," but again, there are problems.

From a computing point of view, the term computer artifact is not customary. Programs appear as formal, symbolic objects, capable of causing computers to exhibit specified behavior. The emphasis is on the development of hardware and software *systems* or *products*, there is little consideration for their embedding in the human world. The term *system* is used in a technical ("hard") sense, referring to a combination of components on different technological levels that, when active, can assume a large but finite number of possible states, brought about by external inputs and the effect of programs. The term *product* emphasizes that software is a sellable commodity. It entails a host of connotations known from industrial production such as division of work, profit, product quality, and the distinction between product development and maintenance. The category relating products with human activity is *usability*. Thus, people communicating or performing their work with the help of computer artifacts are simply users, regardless of their field of expertise, the context of their work and their connection with others.

From a social science point of view, the term computer artifact conveniently integrates the levels of hardware and software and the connection to other technical components required

for a program to become effective, their distinction being of no interest at a use-oriented level of analysis. Social theories provide a conceptual basis for analyzing computer artifacts in the context of human activity, considering them as objects of, means to, or constraints on human communication and work or as constitutive for social change on a large scale. But these theories originate in other fields of interest such as the study of ethnic cultures or of traditional industrial production, they relate to older forms of technology, and offer no specific concepts for the computing realm. Also, they use their own jargon. For example, the terms *artifact* and *human activity* belong to the vocabulary of different social theories. Can they be used outside specific theoretical frameworks?

These observations suggest that we need to work toward a common language, or rather to find ways of deconstructing and reconstructing concepts used in the different disciplines. But beyond terminology, there is the challenge of bridging the gap between world views embodied in the disciplines involved. Computing technology implements the ideal of abstract, objective knowledge, while social theory is faced with the place of this technology in situated and unique human activity unfolding in time.

The formal theory underlying computing is based on foundations in logic, abstract mathematics, and formal linguistics. Also, in the discipline of computing, there are fundamental views equating humans with computers. Even in formulating his basic theoretical concepts, Alan Turing, the most influential thinker inspiring the mathematical science of computation, explained the functioning of his universal machine by likening it to a human being proceeding according to rules. This world view is shaped by the history of formalization in Western culture since the beginning of Greek philosophy and strongly relates to the philosophical program established by Descartes that eventually gave rise to the rationalistic tradition.

Computing emerged at a time when the discussion throughout the humanities was dominated by the rationalistic tradition. Positivistic philosophy, formal approaches to linguistics, behaviorism as a mechanistic approach in psychology, Taylorism in organizations as a way of rationalizing human work, and many other developments provided a scientific environment conducive to extending the central notions of computing into the human and social realm. But while formal theory remains firmly rooted in the rationalistic tradition, the relevance of this tradition to understanding human thinking and activity is subject to hot debate. In the last few decades there has been a tremendous surge of theories challenging its basic assumptions.

Social theory at all levels has rediscovered the situated nature of human activity, shaped by history and tradition and creatively engaged in shaping further development. Not only is the rationalistic tradition unable to account for perspectives related to social relations and power struggles; its assumptions on knowing and acting have also been questioned. The instance of knowing is no longer the individual but the community drawing on a tradition of practice and relying on artifacts. Knowing does not take place “in the head” but is enacted in the body as a whole. Human thinking is no longer seen as essentially governed by rules, but is thought to be rooted in bodily action through images, experience, and intuition. The ideal that all knowledge is explicit, general, static is being given up in favor of assumptions about tacit, situated, dynamic ways of knowing. Strangely enough, computing technology itself and the changes in society triggered by it contribute to shaking the very foundations it was originally built on.

Since software practitioners have to go back and forth between the formal and the social world, a suitable treatment of software practice must comprise both theoretical perspectives. In what follows, I will not argue for combining these perspectives into one common super

theory but for creating a dialogical framework, taking both perspectives into account. Such a framework has to allow for the formal properties of software on the one hand and for the social context of their development and use on the other.

2.3 Levels of Software Practice and Social Reflection

In keeping with the proliferation of computing technology, software practice is all around us. As a first orientation for relating software thinking to social practice I propose to distinguish technology levels according to the complexity of relevant social contexts for software. These levels can be associated with the emergence of technologies in time, but from a current perspective they coexist and often are nested or linked in one system. They will be outlined here in terms of the basic technology, the skills required of software practitioners, the guiding metaphors used in development, and the social context relevant to theoretical reflection.

The basic level is that of *programs* for automating specified computations or algorithms, designed to solve a formal or a real-world problem. Programming relies on a clear separation of the human and computer elements. While this is the oldest form of software practice, it still survives today in the form of automata or agents performing a function as part of an encompassing system. The essential demand is that the program fulfills the functional and technical requirements. The basic skills required are to master the programming language and the programming environment. There is hardly any social context. Machine, system, and automaton are guiding metaphors. The computer is used to automate separable parts of individual work in both scientific and commercial computation. The social reflection involved pertains to likening or contrasting human faculties of individuals with those of computers.

Interactive workplace systems enable a close intertwining of human work steps and computer functions and give rise to choices in design. Beyond personal computing, local networks allow for providing computer support for groups. This is the realm of system design with users. In addition to the skills required for programming, this requires an understanding of human cognitive faculties, of ways and styles in performing and structuring work, and the place of artifacts in human problem solving. On the technical level, interface and local network technology come into play. Questions of usability arise in connection with the need to design the human-computer interface. The relevant social context is human work and communication processes, and the guiding metaphors for computer artifacts are tool and medium. Thus, it is here that we find reflection on computers in the context of human activity.

Enterprise information systems codify structural aspects of organizations. They come with problems of integration and (organizational) standardization on a large scale. Usually it is not a question of developing new systems but of adapting existing systems, so design pertains to how to introduce the system in the organization at hand. Technical challenges lie in using components for tailoring systems to specific needs. The relevant social context is organizational development. Software practitioners are engaged in organizational intervention, being perceived as agents of change. They also have the role of mediators between organizations and vendors. There is less scope for design, because existing structure within organizations provides constraints on choices, and furthermore the introduction of software also promotes standardization across organizational boundaries. The relevant social reflection pertains to technology as upholding structures in organizations.

Computational Infrastructure is the basis for the networking of communities within or across sites. The emphasis shifts from the individual computing artifact to families or landscapes of artifacts, accessible from particular technical platforms and allowing or inhibiting communication and cooperation. Infrastructure is associated on the one hand with technical standardization and compatibility; on the other hand it is closely related to working styles and the culture of communities, which is embodied in shared practices of technology use. A community can be well defined and localized in one physical setting, or it can be spread out geographically and be a loose assembly of people sharing some kind of interest. Associated with infrastructure is the notion of ecology, emphasizing the coevolution of individual components or parts and the close intertwining of the technical and social levels. In keeping with this, there is the notion of "growing" rather than of designing infrastructure.

Networks of humans and computers provide open spaces for opportunities shared globally with unknown participants. The basic technology is the Internet and the World Wide Web, associated with as yet poorly defined ways of Web-engineering methodologies. Technological challenges include security problems, data communication on a large scale, and mass data management. Information technology becomes a cultural technology, comparable to book printing or mass media, and gives rise to profound social changes, making information a commodity and allowing communication to overcome space and time, thus giving rise to globalization. This level of software practice involves developing Web sites and Web applications in keeping with the needs of the emerging information society. Through suitable portals and links, it becomes possible to access a seemingly unlimited world of nodes that provide knowledge and services of various kinds. Social reflection on technology pertains to the nature of networking and the interlinking of human and technical nodes in networks.

As shown, individual theories can be related to different levels of software practice. In what follows the emphasis will not be on any specific level but on how to relate the technical and the social realm.

2.4 Theoretical stands in relating computers and the social realm

In this section we look at social theory as inspiration or guidance for understanding problems arising in software practice and bring out differences between families of theories, as they become relevant to computing. Several metatheoretical distinctions or fields of tension are elaborated so as to show how they become influential in software practice.

Logical-empirical or hermeneutic-dialectic This distinction between families of social theory approaches is made by Nissen in this volume (see chapter 4) and is in keeping with the one made earlier in this chapter. As we have noted, viewing the social world in the spirit of the rationalist tradition creates conditions of uniformity that encourage the use of computing concepts in the social world or the modeling of parts of the social world in computing terms. Also, in the logical-empirical tradition technology development appears to be governed by objective facts or constraints. Adopting a hermeneutic-dialectic viewpoint on the other hand leads to an inherent field of tension and necessitates bridging the gap between two world views in software practice. Many authors who take a critical stand in the hermeneutic-dialectic tradition tend to subordinate the formal/technical world to the social world, so that technology appears plastic and molded according to the interests and values of its owners.

Objective/realist or subjective/constructivist This distinction in social theories is related but not identical to the one made above and has a special significance for software practice. Unlike social reflection, software practice is not content with observing, but is concerned with bringing about change. It is in itself constructive in that it is oriented to developing, introducing, and enhancing computer artifacts. A basic distinction is between approaches that suggest that technological change essentially mirrors and reinforces social structure and views that concentrate on the innovative and inventive character of software practice. This field of tension also relates to relying on ontological argumentation or taking an epistemological stand. Typically, the former will emphasize the constraints resulting from things as they (supposedly) are, while the latter will point to opportunities related to design. These two approaches imply different role models for software practitioners.

Abstract or concrete Social theories relevant to computing differ in terms in their methodological stance. In particular some theories aim at staying close to the situated phenomena as encountered in the field of observation, while others use abstractions for categorizing these phenomena. In connection with software practice, relying on abstract categories is closely related to modeling. Since any program embodies a computational model of the area of interest, modeling is inherent in computing. But the computational model is not necessarily derived from an explicit model of the application area. Social theory approaches providing inspiration for methods in software development will suggest quite different ways of proceeding. On the one hand, explicit modeling creates more distance to the social world and brings about distortions, since the abstract categories used in modeling shape the view of reality represented in the model. On the other hand, explicit modeling is an opportunity to negotiate how the model should be constructed, as well as what features of reality should be represented and how.

Actor-oriented or systemic This distinction relates to how social theories portray the social world. Actor-oriented approaches bring out the scope of influence, highlight the opportunities and commitments of individual or collective actors and clarify the relationships and conflicts between them. Systemic approaches emphasize the integration of actors into a coherent whole and concentrate on emergent systemic properties. In connection with software practice, these theoretical approaches are often perceived as being connected to interests of different groups. While actor-oriented approaches are usually associated with bringing out the interests of the different parties involved and are fundamental to participatory design, systemic approaches are taken to stand for management positions. A new category, particularly relevant for studying social changes around information technology, is the *network*. It can be used with either an actor-oriented or a systemic interpretation and has the potential for mediating between them.

Technology as structure or agent To understand software in social contexts, it is also important to see what status technology is assigned in the theoretical approach. Again, there are two extremes with a field of tension. Is technology considered part of the environment of the social world (and vice versa)? If so, how are they related to one another? Or are technology and the social world seen as integrated into one whole? And if so, what is their mutual dependency and what do they have in common? There are at least two ways of looking at technology in connection with the social world. One approach emphasizes its potential for *reifying structural aspects* of human activity. Here, technology has the essentially passive role of an external memory for human learning and action, freezing routines that become "alive" only in the context of situated use of technology. The other dimension is *agency attributed to*

technology, recognizing that artifacts have their own way of becoming effective. If so, is technical agency taken to be similar to or different from that of human beings?

The reader may use these distinctions and fields of tensions as ways of deconstructing and assessing proposals for understanding software in social contexts. The last field of tension takes us directly to the conceptual framework I am about to present in the rest of this chapter.

3 Developing Autooperational Form

The need for a conceptual framework first became clear when looking closely at the problem of establishing requirements for interactive software systems. These systems involve three types of entities, which have to be understood in different terms: organizations (in terms of tasks), people (in terms of activity), and computers (in terms of functions). Whenever we make design decisions on the functionality and interface of computer-based systems we affect the fields of tension between these entities and we take a stand–often implicitly–on the nature of their relationship. It seemed important to develop a terminology that allowed us to mediate between them without blurring their differences. Later, this conceptual framework was elaborated and expressed in more general terms so as to be applicable to other areas of computing as well.

3.1 Bringing out Operational Form

In what follows, I use the concept of operational form as a key to understanding how human activity can be computerized and how computer artifacts can be embedded in social contexts.

Let us first look at the concept of *operation*, which is established in different domains, all relevant to the discussion here. In organized human activity (ranging from the military, to medicine, to enterprises of different sorts), it refers to proceeding in the situation based on planning with a specified objective. In mathematics, it denotes a computation step in a formalized and interpretation-free manner. In connection with technology, it refers both to the working of machines and to making a machine work. In activity theory, it refers to quasi-automated routines acquired by people through experience with action, the idea being that the levels of action and operation can be analytically separated through scientific observation.

Common features of the concept of operation as used in these domains are:

- Operations are rooted in repeated human action of individuals or groups.
- Fundamental to operations is the separation of description and performance.
- The description of operations enables well-defined ways of proceeding that can be taught, planned, and enforced.
- The performance of operations is embedded in situated human activity.

In what follows, we will concentrate on developing a concept of operation suitable for the realm of computing, and focus on the interplay of description and performance already implied in other uses of the term.

The description of an operation requires an observer. This does not necessarily imply an individual carrying out the observation ad hoc, more often there is a tradition of description used as a reference for common action. Descriptions allow for disconnecting operations from human beings acting in situations in a unique way.

Through description operations are given a name, they are characterized and distinguished from other operations or actions. Individual operations are described in terms of material or abstract objects to be dealt with, of prerequisites needed for the operation to be performed, of effects to be obtained, of rules that govern the operation, and of means to be used. Descriptions are always made for a purpose. They can be used for teaching, planning and coordinating human activity

Operational form results from connecting the descriptions of individual operations in terms of temporal, logical, and causal relations. This connection again needs an observer. Operational form can itself be considered an operation on a higher level, thereby leading to complex, interdependent structures of operational form.

The performance of described operations requires tailoring situated activity with a view to fitting the conditions and constraints specified in the description. Thus, operational form does not specify the activity itself, but the demands on activity in terms of well-defined results.

The emergence of operational form is essential for all organized human activity in our culture, so much so that "operational thinking" evokes the basic paradigm of the Western way of dealing with the world. Computing relates to operational thinking in a particularly poignant way:

- Operational thinking in mathematics, philosophy and language has led to the notion of algorithm, which is fundamental to expressing operational form through programming.

- Operational thinking in science and technology has given rise to ways of understanding nature and to the development of machines on the basis of specified operational form.

- Operational thinking in organizations is at the root of professional and standards in diverse fields of human practice.

Before shedding further light on how computing connects the world of human practice, formal method and technology through operational form, we need to look at artifacts making operational form available to human activity.

Description of operational form through symbolic artifacts Though operational form can be traced in oral tradition, it becomes vastly more effective when codified in writing. Thus it can be transferred through symbolic artifacts across space and time.

Operational form in situated action Predefined operational form is interpreted in situated action, thereby bridging the gap between situational needs and specified objectives and constraints. Also, operational form is constantly refined and revised through human action. The more generally applicable operational form is intended to be, the more its description will need to be unambiguous and interpretation-free, and the more emphasis is likely to be placed on making it unchangeable.

Performance of operations by technical artifacts If operations and their connections are described in an interpretation-free manner by specifying only formal properties (such as input and output, rules for performing the operation, and so on) they can be mechanized, and therefore executed by technical artifacts like machines. Unlike human actors, technical artifacts as nonhuman agents merely execute operations; they only interpret them to the extent that this interpretation itself is specified as operational form.

Artifacts opening spaces of opportunities for situated action On the one hand operational form is reified through artifacts; on the other, when used in situated human activity, the same artifacts give rise to new opportunities, conditions and constraints for human activity. The result can be perceived in cyclical terms with situated activity leading to operational form, this being reified in artifacts, these opening new spaces of opportunities, which unfold through situated action, again leading to operational form, and so on.

Thus, operational form is associated both with symbolic artifacts through description and with technical artifacts through performance. The distinctive feature of software is that it is symbolic and technical at the same time. Thus, software can be studied along both dimensions suggested in the previous section. As symbolic artifact it has the potential of codifying operational form, which is taken to correspond to the part of the social, technical or natural world of interest. As a technical artifact, it has the potential to execute formally specified operations. Some important steps are still needed, however, to tailor the general arguments proposed so far to the realm of computing:

- Computing is concerned with informational operations, dealing with informational objects and associated with prerequisites, effects, conditions, and rules that can be specified in terms of information.

- Computing rests on making operational form explicit and decontextualized. That is, where operational form is already found, for example in social contexts, it needs to be specified in detail. Otherwise it needs to be invented so as to fit or simulate the area of concern.

To sum it up, I suggest that the software computer as a technical agent can replace a human being in terms of operations, not in terms of actions. The question then is, what is new about a computer as opposed to other machines executing operational form? This will be pursued in the next section by showing that computer artifacts are not tied to fixed operational form but are instructed by modeled operational form embodied in computer programs.

3.2 Operational (Re-)construction and Autooperational Form

In the field of computing, method tends to refer to prescriptions like top-down or bottom-up design procedures or the use of certain higher-level languages and diagrams, which may be incorporated in a comprehensive approach to software development. Instead we will reflect on the basic paradigms implicit in software development so as to meet the needs of an application. The crucial step to be taken is to *model* operational form. This constructive activity relies on modeling concepts and on ways of applying these concepts to the area of interest. The overall process of modeling operational form will be called *operational (re-)construction*. We start from the following assumptions:

- In connection with computing, the relevant area of interest is considered in terms of operational form.

- Modeling, a constructive process, is inherent in computing.

- Computing is concerned with reproducing existing operational form and inventing new operational form, the boundary between these two concerns being indistinct.

Operational form can be (re-)constructed in different ways or styles, which have been consolidated in modeling concepts belonging to so-called programming paradigms. The *procedural style* rests on modeling operational form through standardized procedures or functions operating on collections of informational entities. The *object-oriented style* views informational entities or objects as the starting point for modeling, and provides a repertoire of connected operations associated with objects that can be combined in flexible ways. The *constraint-oriented style* is only concerned with specifying the desired effect of operations on informational entities and leaves the operations themselves open.

Irrespective of the modeling style adopted, operational reconstruction implies specific ways of dealing with the area of interest.

Building informational objects and operations This involves deciding what should be included in the model, how it should be represented, and how it should be connected to the surrounding world in informational terms. Relevant objects may be abstract or material things in the natural or social world or in a formal universe of discourse. Operations need to be chosen so as to bring about observable changes by natural or technical processes or by human activity. Where computing is used to deal with problems of the real world, informational objects and operations are often used to substitute for their material counterparts. Considering something informational means making explicit an informational content that so far has been implied, often replacing sensual or bodily ways of dealing with the world with abstract intellectual ones. This in turn is the basis for deciding which phenomena and events should be used as data to represent the informational content and attributes of objects and operations.

Specifying objects and operations in discrete terms Computing invariably has to do with identifying and standardizing discrete operational steps and grouping them into complex formalized procedures, which give rise to algorithmic structures to be executed on a computer. Objects need to be specified in terms of discrete attributes. Both space and time are modeled in discrete terms. Classifications are used to separate related cases in the area of concern or to distinguish states. Thus, discretizing is not only implied by the digital nature of technology but inherent in the process of modeling operational form.

Integrating objects and operations into a system In computing two levels of systems formation need to be distinguished and related: the *computer system* consisting of hardware and software parts, and the *referent system*, consisting of the part of the technical, natural, or social world that is modeled, affected, manipulated, or simulated by the computer system. Forming a computer system is inherent in computing. On the software level it consists of the informational objects and operations obtained through modeling and their connection to the technical environment through technical signals. The notion of a referent system is less obvious. Clearly, a referent system can and must be considered when computer systems are used to simulate or control natural or technical processes. But where software is embedded in so-

cial contexts, it is a question of the social theory adopted, and more specifically, whether the social world is considered as a system and if so, how.

The result of operational (re-)construction is an operational model of the area of interest that can be reified through description in a suitable symbolic form. To bring about the operation of a computer, such descriptions must be expressed in programming language, allowing for the unambiguous statement of operational structures (algorithms and data structures) corresponding to the operational model, thus leading to *autooperational form*. The term *autooperational* is deliberately chosen in analogy to the term *automobile* for car, denoting a technical device moving on its own. The suggestion is that the computer is a device operating on its own, energized by electricity of course, instructed by operational form described in programs, and triggered by situated human activity in use.

4. Embedding Autooperational Form

Embedding autooperational form in social contexts has a structure and an agency dimension. On the one hand, autooperational form is interpreted in human activity; on the other, it becomes effective when executed on a computer. Note that there is a clear distinction between *actors* and *agents* implied, the former being reserved exclusively for humans, while the latter is applied to non human technical agents.

The common feature of all computer operation is that it is based on a model representing a system of discrete, interacting informational elements. How these are modeled, and how this model is related to the social context of use, accounts for the tremendous variety of existing computer artifacts and is the central issue in design. Since all operational form rests on reductionist models of entities carrying informational attributes, the modeling concepts shape the way we think about the world. The relation of formal models to reality is not given but constructed in the design process. This involves deciding which aspects are included in a model and how the model is to become operational in use.

Perhaps the most important question is the following: What kinds of interaction between human actors and autooperational agents do we allow for? We cannot discuss this without taking a stand of our own, referring to our basic values. Since we are manipulating the scope for humans to take responsibility in situated action, serious ethical considerations are at stake. In the rest of this section, we touch on some fundamental concerns that relate to embedding autooperational form.

Design and scopes of choice A basic concern in design is modeling operational form with a view to human use. This requires that the entities to be modeled are perceived in terms of the social context and that the human-computer interaction fits the needs of the situated activity. Design rests on the possibility of making *choices*. And through design, choice scopes of users are influenced. For example, if computer artifacts are designed as tools or media, the idea is usually to increase the scope of choice for human users. This is not so obvious in workflow systems, where the idea may be more to constrain the scope of choice available to users. Also, when we consider computers as artifacts, they are designed to mediate human activity. Here, autooperational form is inevitably interpreted in situated action, leading to creative forms of computer use. On the other hand, designing software as intelligent agents is associated with the idea of allowing them rights and a scope of operation comparable to that of humans. Thus, design influences but does not determine the available options.

Autonomy versus Rules Since software design rests on bringing out operational form, it is intimately connected with rules. There is an entity responsible to set up rules: a human who makes or identifies rules (for example the software developer complying with the wishes of the customer), an entity that imposes rules, often a non human agent (the computer controlling work processes or technical systems), and an entity that carries out rules, either human or technical. Imposing automated formalized procedures through computers brings about a new quality. While in general, the rules imposed on human activity have to be sufficiently clear and unambiguous for humans to follow, here they must be formulated in machine-interpretable terms. While humans interpret rules as they apply to the needs of the specific situation, programs always operate according to their predefined model. And while humans tend to associate rules with exceptions, computers do not.

Automating or informing To allow for the use of computer artifacts in different situations, the designer must have a mental picture of the class of possible situations and a sufficiently rich understanding to allow for any potentially relevant activity at any time. There are two basic options for embedding formalized procedures in the richness of human situations. One is to accommodate all possible use situations in a formal model and to automate a set of rules for how to proceed according to this model. The other is to leave the use context open and to offer a repertoire of resources to use in self-organizing work, thus contributing to making the situated activity more informed.

Ways how autooperational form becomes effective In use, autooperational form becomes effective in different ways. For example, it can be supportive or controlling, informative or instructive, simulative or directly affecting reality. I do not know a good classification scheme, but we need to work in this direction, as the proliferation of computing technology proceeds further and further. Perhaps we need something akin to the speech act theory in the philosophy of language: an operational model effect theory that would provide us with conceptual categories for discussing how computers interfere with human activity in situated use.

Populating the world with autooperational agents In my view, computer programs in operation belong to a novel kind of entities. If I were Popperian interested in ontology, I would call for a "fourth world" consisting of human-made autooperational artifacts. Staying close to software practice, I would rather point out that by making a host of interconnected software systems of different types available, designers enable complex styles of human-machine interaction where the full consequences of human action triggering computer operation, and the mutual rights of human actors and technical agents in networks are only clear to specialists. This calls for an increased concern with security and transparency.

Viewing computing in terms of making autooperational form available does not, in itself, suggest a way of dealing with these ethical issues, but it does make it possible to discuss them appropriately so as to raise awareness of the problems involved.

5. Autooperational Form as a Way of Viewing Software in Social Contexts

As this chapter comes to a close, the conceptual framework offered here needs to be related to the different levels of software practice distinguished and assessed in terms of the theoretical proposed on section 2.

The idea of developing and embedding autooperational form is relevant for all levels of software practice distinguished according to their social contexts. But of course, operational form comes about in different ways, and the design scope is not always the same. In particular, complex social and technical contexts are associated with standardization, usually imposing limits on design. Programs describe and execute operations on informational objects in a largely decontextualized way. Interactive systems reflect operational form pertaining to how individual work steps can be interrelated with computer functions and how cooperation and coordination can be carried out in groups. Enterprise information systems codify operational form underlying, for example, workflows and business processes. Infrastructure embodies operational form related to combining the use of different artifacts and components. Networks provide operational form for relating different human actors and technical agents.

Although the position taken here does not claim the status of a social theory, it can be located in the territory of social reflection. It belongs to the hermeneutic-dialectic tradition, bringing out, as it does, the complementarity of situated action unfolding in time in unique ways and operational form, stating general decontextualized principles. It is constructivist in that it portrays software development as a creative process in terms of the scope of choice available, but it does not blind itself to existing structures that are expressed in terms of operational form. It is certainly abstract, since it provides analytical categories for connecting computer artifacts to human activity. Operational form lends itself to an actor-oriented or a systemic view, depending on whether the emphasis is on how it is interpreted in human activity or on how it is constitutive for integrating the elements of a system into a whole.

The main purpose of the framework, however, is to contribute to an understanding of technology as both structure and agency. The concepts offered here make it possible to discuss these dimensions concretely as they relate to software. On the one hand, operational form is a specific way of codifying structure that can subsequently be interpreted in human activity. On the other hand, understanding technical agency in terms of operation rather than in terms of action offers a way of relating human and technical agency that avoids the strong symmetry implied in regarding the two as equivalent.

My aim in this chapter has not primarily been to urge adoption of the terminology proposed but to increase awareness of the distinctions implied. For software practitioners this discussion should provide a better idea of the social context, as it relates to what is modeled in computer programs. Social thinkers reflecting about software practice may acquire a richer understanding of the way software relates to the social world. Above all, I would like to promote a view of the computing discipline that makes it possible to address the connection between the formal, the social and the technical worlds in a suitable way.

Notes

The conceptual framework reported here was deeply influenced by several doctoral theses that I had the privilege of supervising. In particular, I have been inspired by the work of Reinhard Keil-Slawik, Fanny-Michaela Reisin, Reinhard Budde and Heinz Züllighoven, Guido Gryczan, Ralf Klischewski, and Yvonne Dittrich, listed here in chronological order from earliest to most recent dissertation projects.

The reflection on how to relate social thinking and software practice has benefited greatly from discussions with Wolf-Gideon Bleek, Ralf Klischewski, Bernd Pape, and Ingrid Wetzel, all associated with the groups for Applied and Social Informatics and Software Engineering at the University of Hamburg, while I was writing this chapter.

1 The conceptual framework presented here was published only in German so far, first as a contribution to a workshop on the philosophy of science (Floyd, C. 1997. Autooperationale Form und situiertes Handeln. In C. Hubig, ed., *Cognitio Humana–Dynamik des Wissens und der Werte*. Berlin: Akademie Verlag, 237-252) and then as part of a study text for students in informatics (Floyd, C. and H. Klaeren 1999. *Informatik als Praxis und Wissenschaft*. Tübinger Studientexte Informatik und Gesellschaft. Tübingen: University of Tübingen). For those who are interested and read German, I would like to point out that this study text goes well beyond the scope of the present chapter. It demonstrates connections to the origin and history of computing and discusses the contributions of many authors who have dealt with issues relating to my line of argument.