2000

On Some Epistemological Problems of Software Engineering

Peter Schefe
Universitat Hamburg, schefe@informatik.uni-hamburg.de

On some epistemological problems of Software Engineering

Peter Schefe
Fachbereich Informatik
Universität Hamburg
Vogt-Kölln-Str. 30
D-22527 Hamburg
schefe@informatik.uni-hamburg.de

Abstract

The paper addresses some misconceptions of Software Engineering, requirements analysis and modelling in particular, due to underlying epistemological flaws., e.g. the believe that the system analyst's task be simlar to that of a natural scientist's. The fundamental issues, constitution of objects and signs, conceptualization and definability, are discussed. It comes out that the paradoxical situation of software engineering is having to formalize what cannot be formalized. This is reflected in the fuzzy notion of 'model' in general as well as in the epistemological presumptions of 'object oriented modelling' in particular. The paradigm of 'objective modelling' has to be replaced by a paradigm of 'purposive description' shifting the focus of Software Engineering research to non-formal methodologies.

1. Introduction

A definition of Software Engineering (SE in the sequel) such as given by J. McDermid [..]:

> [...] a science and art of specifying, designing, implementing and evolving – with economy, timeliness and elegance – programs, documentation and operating procedures whereby computers can be made useful to man. ([8], p. 2)

exhibits the typical engineer‘s concept of her discipline. The twofold characterization of “science and art“ can be traced back to Aristotle‘s dichotomy of theoretical knowledge and practical ability. It goes unnoticed that the generic part of this definition, "science", and the specific part, "programs, documentation, and operating procedures", are implying epistemological presuppositions contradicting each other.

As to the first issue, the foundation of SE in science presumes that the software engineer is in the epistemological position of a scientist:

> The task of the system analyst may indeed be considered to be essentially the same as that of the scientist, in that it involves constructing an abstract mathematical model of a real world domain of phenomena. ([9], p. 3)

This comparison gives rise to a deep epistemological misconception, especially, as to the role of mathematics in the building of a software "model of the real world".

Science uses mathematics to conceive of natural laws allowing for explaining and predicting phenomena in the physical world represented as measurements. Thus, mathematics provides with an abstract modelling facility, e. g., the model of Newtonian mechanics may be used to program the control of rockets, say. Obviously, it is not the task of software engineers to develop such models, yet they have to understand them to some degree, in order to translate them into programs.

The contradiction becomes apparent, when "familiar applications" - which can do without such mathematical models - are taken into consideration:

> Taking the ‘scientific view‘, the specification, especially that part which specifies functional behaviour, is best written in a formal language, that is a language with precisely defined semantics. Such languages in this context normally have a semantics defined in mathematical terms. ([9], p. 3)

Here, the role of mathematics in specifying sofware ‘models‘ is quite different from the role it takes in natural sciences such as physics: it is not used for prediction and explanation of real world phenomena, instead, it provides with the formal semantics of the *language* in which the software ‘model‘ is written. Fig. 1, the Software Engineering Triangle, shows the problem constellation more obvious in familiar applications.

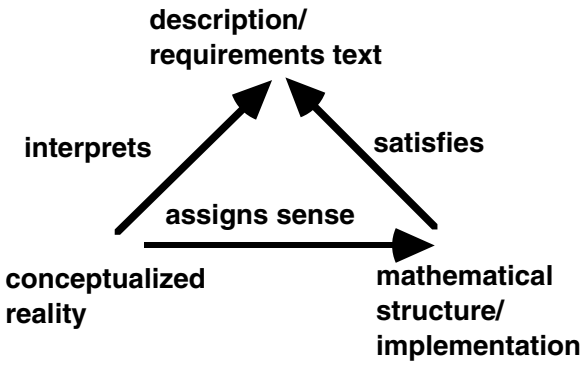

Fig. 1 Software Engineering Triangle

The ‘model‘s‘ *text* occurs in two roles: First, it is a theory satisfied by a mathematical model implementing the specification. Second, it is a description interpreted by real world concepts giving rise to a mediated relationship between real world and the software system that may be called "assignment of sense". This assigment cannot be accounted for formally. To "satisfy" the formal specification and to "satisfy" the informal requirements are epistemologically quite different issues.

As a consequence, the work of the system analyst cannot be viewed in parallel to that of the scientist‘s work. The former has to construct a text to desribe, the latter hypothesizes or uses a law to explain or to predict. There are both epistemological and practical consequences. As to the latter, I would like to quote McDermid once more:

> In practice it is very rare for problem domain theories to be re-used and this is an area where our current practices exacerbate the inherent problem.([9], p. 3)

This "inherent problem" is not grounded in poor programming techniques, but in the epistemological situation of SE. Its goal is not to arrive at universally valid laws, but the provision of means that are adaptable to changing purposes. The problem is a philosophical one. It raises several philosophical issues I will elaborate on in the sequel: What are objects? What is software? What can be represented in software? What can be defined and formalized? What are ‘objects‘ in ‘object-oriented‘ systems about? What is a (software) model? These questions are related to some general issues in ontology, epistemology, and philosophy of language.

2. Objects

"Object-oriented" analysis, design and programming have rendered the notion of "object" a central issue of SE. It appears to be a common conjecture that this methodology will be appropriate for constructing software as isomorphic models of the real world. As Coad and Jourdon [1999] put it, naively:

> Object-oriented analysis is based upon concepts that we first learned in kindergarten: objects and attributes, classes and members, wholes and parts.[1]

What are objects? There are two basic answers to this question:

- Objects are liguistic-logical entities
- Objects are real world phenomena

The first approach is due to G. Frege[3] in the first place. The "Tractatus" of L. Wittgenstein[19] gives the explanation:

> Two objects[...] can be distinguished only by their being different objects[2]

and W.V.O. Quine provides with the famous formulation:

> To be is to be the value of a bound variable.([13], p. 103)

This formal notion of object cannot account for phenomena, namely material things such as persons and physical objects. Whilst the Tractatus declared formal objects to be the substance of the world, the Aristotelian tradition drawn upon by Madsen et al. [1993] considers "substance" to be some substrate giving objects their identity:

> Phenomena like 'Sokrates', 'Mount Everest', a medical record, and a memory cell in a computer are all examples of phenomena which have substance. Substance is characterized by a certain volume and a unique location in time and space. A major aspect of substance is *identity.* Two pieces of substance have the same identity only if they are the same substance. (p 294)

There are difficulties as to this "definition": How is the "identity" of Sokrates bound to a certain volume? How could we measure his volume without identifying him before? Doesn't he change his volume continually? The last statement, then, shows the circularity of argument that makes this explanation worthless.

We need a principle of concrete individuation and suitable conditions of identity. A plausible principle has been proposed by P. Strawson [17]: Objects, e. g. 'Sokrates' and 'Mount Everest', can be referred to using a real system of reference, the system of temporal and spatial relations in which every individual is related to every other in an unambiguous way. Together with recognizability, this frame of reference enables objects to

---

[1] Coad und Jourdon (1990), quoted in [19], p. 2

[2] 2.0233 Zwei Gegenstände [...] sind[...] von einander nur dadurch unterschieden, daß sie verschieden sind.

be identified. By this principle, physical objects gain a central role in the identification of individuals in general. Processes, properties, and states are individuals that can be identified only secondarily, that is to say, with respect to some primarily identifiable entity. This bears some evidence for the appropriateness of "object-oriented modelling".

But how do the two notions of "object" mentioned comply with each other? Formal objects are pure particulars; they lack identity; they do not belong to the physical world; they can only be distinguished from each other in the Tractatus' way. A reflection of this dichotomy is the software distinction of "values" being non-locatable abstract entities and identifiable "objects" that can be and have to be accessed at *physical* locations.

This may lead up to the question "What is software"? Is it an abstract entity such as a number or a physical object such as a magnetic tape? A. Turing [18] was the first stipulating that computation must be mechanizable, i.e., realizable in a physical medium. He indicated two ways to do so, first, to construct a mechanical device (hardware), second, to imprint marks on some physical carrier (software). This has some interesting philosophical consequences pertaining to SE: software and hardware are equivalent; both software and hardware are participating in the world of pure particulars as well as in the world of individuals. Put in another way, there is no *ontological* distinction of software and hardware such as conjectured by some philosophers of Artificial Intelligence[1], and it is mistaken to address software as a pure mathematical entity [2]. This result corroborates our former statement that natural sciences are not the appropriate model for SE: the physical patterns of a magnetic tape marked by a writing device cannot be deciphered using methods of physics, but only by another technical, i. e., man-made purposeful device. Software construction is not "physical modelling", as E. Madsen, B. Möller-Pedersen, and K. Nygaard[7] have in mind, but purposive construction of abstract objects realizable in an arbitrary physical medium.

A last issue of "objects" is in order. The notion of object implies relativity. There is no absolute condition of what counts as an object and what exactly an object amounts to, e.g. what exactly belongs to 'Mount Everest', whether a broken chair is still a chair etc. As H. Putnam [12] has pointed out, it is a matter of degree what factual and what conventional aspects are determining our grasp of an object. To emphasize it: "modelling" and identifying "objects" in our world is not a natural scientist's task, but a task of somebody capable of understanding and communicating human conventions, needs and purposes.

3. Representations

As has been mentioned, it appears to be a common understanding in SE that objects of the real world are "represented" or "modelled" by software isomorphically, ideally at least, says D. A. Stokes[16]:

> Ideally, we would use languages which had a single interpretation, i. e. the language would be 'isomorphic with' the real world; each part (e. g. word) of the language would refer to only a single 'thing' in the real world. Such languages would allow the analyst to write system descriptions that would be entirely unambiguous, and which could be only correctly understood by both the customer/user and the designer/analyst. (p. 16/5)

Once more, this reminds of the Tractatus' conception of object and sign as a "model" or "image" of the former. These notions are reflected in the formal semantics concept of Computer Science: a formal association (morphism) of formal objects. The meaning of a formal term is a formal object. This concept of sign and meaning is rooted in deduction establishing pure formal erquivalence relationships [7]. Meanings, however, can only be hypothesised risking misinterpretation. This is not a deficiency of particular human communicative means, but a necessary constituent enabling flexibility and change.

The formal concept of meaning fosters the belief in representation as isomorphic modelling. There are even inherent drawbacks, however. The double role of formal objects gives rise to contradictions, as has been explained by K. Gödel and others. Only if used in restrictive ways, it allows for any level of formal abstraction used in SE for construction and interpretation of "modelling" languages. This restrictive notion of meaning looses its usefulness if transferred to language semantics in general; Madsen et al. are supplying an example involuntarily:

> The programming process involves identification of relevant concepts and phenomena in the referent system and *representation* of these concepts and phenomena in the model system. (p. 292, emphasis by P.S.)

Problems due to their concept of 'representation' may be explained using the following examples provided by the authors:

> The balance property of an Account is an example of a property that cannot be *represented* directly in BETA. Balance is an example of a measurable property, and the substance of the integer part-object does not correspond to a phenomenon in the *referent system*. (my emphasis - P.S.)

But what, then, does a balance of "1000 Euro", say, represent? Nothing? If this makes sense at all, its inappropriateness is grounded in several misconceptions, especially in the notions of "substance" (see above) and "representation" as a correspondence relation, and even of

[1] [1], p. 210: 'The powers of this *virtual machine* vastly enhance the underlying powers of the *organic hardware* on which it runs[...]"

"object". Similarly:

> Consider next the *speed* of a vehicle. This is another property that does not have substance. It is, however, an observable and measurable property, by for example the car's speedometer or the police radar, and the measuring devices do have substance. (p. 312)

Hence, it follows that "speed" (as well as balance) is not an object, that is to say, it is not part of the real world - only speedometers are! As a consequence, Madsen et al. have to state:

> *BETA elements which are not representative are called* non-representative. (p. 312)

What does it mean: "Sokrates is represented by a computer program"? Is there a physical duplicate? Is Sokrates simulated? Obviously not. Sokrates is not "represented" as a physical object, instead, there is a linguistic description 'modelling' the logical structure of a certain conceptual grasp of Sokrates. As a consequence, it makes no sense to look for physical elements corresponding to the syntactic elements of the description. It is mistaken to declare some syntactic elements as "representative", others not, because there be no obvious primarily identifiable objects in the refenrence system. Reference is not an isomorphic relationship between descriptions and their subjects.

The meaning of language terms is conventional and relative, so is the meaning of software representations. The construction and use of software are grounded in purposive action, and the "world" it refers to is to a considerable degree a reflection of the conventional commitment to these purposes.

4. Concepts, definitions, and classes

Concepts are the means for the epistemological "grasp" of our world. The "definition":

> A concept is a generalized idea of a collection of phenomena, based on knowledge of common properties of instances in the collection. ([7], p. 297)

originating from Aristotle seems to be popular in computer science because of its similarity with the notion of formal abstraction. Unfortunately, it does not work for concepts of natural kinds. What would be the generalized idea of a dog? What colour , which size would it have? A typical dog would not be what a universal is conveying to its user. It is closely entangled with the identification of particulars. As Strawson puts it:

> For what constitutes our grasp of the general criterion of identity for men, or for that of horses, or of planets or storms or buildings or battles – or indeed for any kind of particulars – is precisely our grasp of the general concepts under which they respectively fall or (what comes to the same thing) our grasp of the senses of the general terms for the kinds in question.[1]

According to Strawson [17], there are three kinds of universals, feature, sortal, and characterizing universals. I will not comment on the first kind. The use of sortal universals is closely related to our criterion of identity for individuals. Only if the meaning of these *categories* is understood, meaningful discourse pertaining to objects becomes possible, i. e., that characterizing universals can be ascribed successfully. For example, "Sokrates is human" is the presupposition to "Sokrates ist wise". The former is tautological (if not used for introduction), i. e., it is not subject to refutation by history, whilst the latter may be. The examples are related to well-known dichotomies, among others, of analytical and synthetical knowledge, of closed and open concepts, of necessary and contingent properties.

Similarly, to the distinction of defining and characteristic properties of concepts. Can categories be defined? Yes, if not, they would not be programmable, is the answer of SE. This gives rise to the central dilemma of the discipline, as Stokes puts it:

> We need to force synthetic reasoning to be less subjective to bring the statement of requirements into an (almost) analytical framework. If we wish to use formal languages then the problem of misinterpretation must be reduced. How we might achieve this, perhaps by 'narrowing the bandwidth of the observations' that we must make to a minimum, and thereby controlling the degree of misinterpretation, will be discussed later. (p. 16/6)

The 'solution' seems clear: knowledge captured by empirical concepts has to be *forced* into definitions that 'minimize' (i .e. 'zero') misinterpretation. However, this is simply impossible. Stokes' fundamental mistake is the believe that formal definitions implying absolute decidability of membership are a means of precization of empirical concepts. W. V. O. Quine [14] has argued that definitions of empirical concepts are still dependent on prior synonymities that cannot be forced into a closed equivalence relationship. Amazingly, Stokes himself admits:

---

[1] P. Strawson : Two Conceptions of Philosophy. (1990) quoted in [5], p. 191, footnote 27.

> One further fundamental difficulty of RML (an algebraic specification system - P. S.) is the relationship between the world, the model and the system. How can we ensure that the model correctly represents the relevant issues of the real world? (p. 16/19)

Then, he resorts again to a formal method: verification against a formal "enviromental model":

> This would be useful because validation would become more analytical, and less subjective. (p. 16/6)

thus closing his circle of argument. Madsen et al., using the dichotomy of "Aristotelian" (= analytical) and "prototypical" (= synthetical) concept are running into the same trap, when stating on the one hand:

> The realized concepts in the model have to be Aristotelian. Part of the modelling function is thus concerned with giving prototypical concepts an Aristotelian interpretation.

and admitting on the other one:

> This will often make the resulting computer system appear inflexible to the user. (p. 318)

thus exhibiting the same misconception. This is not the solution to the dilemma that real world concepts are not formal entities. Hence, formal reconstructions of real world concepts have to be considered open. Their definitions can only be partial., and, thus, open to change. Whether a concept is "well-defined" in this sense, depends on its usefulness. Even SE concepts such as "system design" and "implementation" are not "objective" in the sense that their extensions could be defined exactly once and for ever. The differentiation is *useful*, as it accounts for different kinds of mistakes.

A special look on 'object-oriented' methodology may clarify the issue. It appears to be a commonality among object-oriented people that:

> ...it is natural in the sense that the design pieces are closely identified with the real world concepts which they model. ([6], p. 41)

Beyond that, it is asserted that class hierarchies do 'model' conceptual hierarchies. A simple example refutes the conjecture that class defining hierarchies are conceptually generalisation/specialization hierarchies. The class of points in the plane is 'modelled' by two 'attributes', the coordinates. A 'subclass' can be derived by adding a third attribute giving the 'model' of a point in three-dimensional space. As the resulting set of instances is not a subset of the set of instances of the 'superclass', the conjectured relationship does not hold: the set of three-dimensional points is not a subset of the set of two-dimensional points. This does not preclude that class hierarchies can be constructed in analogy with conceptual relationships to some degree. Consider the example due to [7]:

```
Reservation:
        Date:...
        Customer:..
FligthReservation: Reservation
        ReservedFlight:...
        ReservedSeat:...
TrainReservation: Reservation
        ReservedTrain:...
        ReservedCarriage:...
        ReservedSeat:...
```

The conceptual relationship of "Reservation" and the two subclasses is indeed a subconcept relationship. However, this is not what is 'modelled' by the classes! In conceptual hierarchies, defining properties are needed for subclassing, "sort of vehicle" would do in the example above. In object-oriented subclassing, descriptive entities are added re-using the description of the superclass. The set-theoretical semantics is not applicable: the instances of "TrainReservation" *are not* a subset of the instances of "Reservation", and "Reservation" is an so-called abstract class with no instances at all!

What, then, are obeject-oriented class hierarchies about? First, class hierarchies are *description* hierarchies tied together by the purely syntactic mechanisms of factorization and extension, respectively, addressing aggregates of descriptive attributes and methods common to all instances. The semantic aspect of subclassing is given by the conformance relation holding among class and superclass, that is to say: an instance of some class can stand for ("is a") an instsnace of its superclass, if and because it shares the syntactic and static attributes of its predecessors. This is, certainly, not a conceptual, but a (software-)technical feature of object-oriented inheritance systems.

Their usefulness, and perhaps their superiorty to other methodologies, is not grounded in - contradicting the term - 'modelling' real world *objects*, but in reconstructing *purposive action* grouped around some material. The following example adapted from [11] is illustrative in this respect, the actions of filing, sorting, accessing etc. material in different kinds of containers:

```
File:
    Insert:...
    Sort:...
    Select:...
    Delete:...
```

The actions that constitute the essence of a file are abstract as long as the implementation is not known. Stacks and folders behave differently, and they may add more specific actions, e.g. "Label" or "Spread-out". Subclassing may be concretization or implementation, not just specialization.

Re-use is one of its benefits, program structuring and modularity are the classical issues addressed, too. As to "modelling", the strength of object-oriented systems is not to describe real world concepts or objects, but to formally reconstruct actions constituting patterns of behaviour of some material to be differentiated into queries and updates. If these "models" are implemented on a real machine, their intended purpose must become obvious to their users. The description of methods associated with a certain (mostly abstract) material, not the description of physical objects makes up the strength of this methodology.

5. Models

SE has tried to address the "inherent problem" of requirement anlysis and specification by methods of prototyping. Unfortunately, the concept of prototype is closely related to the concept of 'model' in which the epistemological misconceptions and fallacies of SE crystallize. One source of difficulty has its origin in the confusion of "model for" and "model of". B. Monahan and R. Shaw conceive of the "engineer's notion of 'model'":

> This is a *prototype* construction, smaller in scale than the real thing, but useful for testing out ideas and checking specific calculations (e.g. the use of wind tunnel models in aerodynamic design). (p. 21/4, my emphasis– P. S.)

A prototype is a 'model for' something to be constructed, a wind tunnel model, contrarily, is a 'model of' something to be analysed.

Furthermore, there is the adoption of a naive 'natural science view': software as a 'mathematical model' for prediction and explanation:

> This is a mathematical theory in which the more fundamental aspects of the system are formulated and discussed. This is useful in making predictions and in drawing conclusions about the system by the use of logical inference. (p. 21/4)

Once more the same confusion: in natural sciences, a mathematical model is a ‘model of", whilst a formal software specification is a 'model for' the system to be implemented. For the former, the relationship between 'model' and its target is synthetical, for the latter, it is purely analytical.

There is at least a third - well established - notion of model, the semantic model. Each consistent theory has a 'model' by definition, a structure that satisfies it. As the system implementing the specification is a semantic model of this theory, we get another source of confusion. Monahan and Shaw appear to make a virtue out of necessity:

> Finally, it should also be clear that model-based specifications involve some of the characteristics of each of the notions of 'model‘ mentioned above. (p. 21/15)

The better alternative would be to avoid the term "model" altogether, and to use unambiguos terms instead, "description" and "prescription", e. g., to differentiate the "of-" and "for-"roles, and "explorative/experimental version" to indicate the status of an implementation.

The examples given by the authors (see below) clearly show that "models" are formal descriptions of some logically conceived structure of organizational reality. The formal specification language Z, a logic-oriented system augmented with set-theoretic constructs, provides the impression of a "model-building approach", i.e:

> The objects are then given existence by building them as compound strcutures, using the more fundamental objects as a base material. (p. 21/4)

The example below exhibits conventional wisdom:

> 2. *Employs*. A relation will be used to model the link between companies and employed people[...] This relationship models the link between the entities **COMPANY** and **PERSON**. The domain of the relationship models the entity set **COMPANY** and the pairs (company, person) models the entity set **PERSON** of those people employed by companies.
> 3. *Vacancies*. A partial function between companies and the number of vacancies each has [...] (p. 21/10)

This is a formal abstraction to be interpreted by the clients as well as by the implementors. However, can the former understand the consequences? What does it mean: **COMPANY** is a "model" of a company and the set of tuples called *Employs* a "model" of its relationship to people? Is this the 'precization' they want? Can, for example, people only be employed, if there are *Vacancies*?

Formal methods do not prevent desasters, as is suggested even by risk-experienced P. Neumann:

> Specification of abstractions and their relationships are also sources of desasters.[1]

Nevertheless, he misses the point that formal methods are not the lesson to be learned:

> In the absence of stringent analytical techniques (e.g. formal methods), oversimplification tends to be discovered only after failures were observed.

To the contrary. As the SE dilemma cannot be solved

---

[1] P. Neumann: Risks of Easy Answers. In: CACM 38 (1995), p. 130

simply by applying "stringent" formal methods, SE has to shift the focus of research and education to non-formal methodologies.

6. Conclusion

The discussion above let stand out some fundamental conceptual problems of SE.

The - perhaps most momentous - misconception of SE is to classify it as a natural science.

• Not natural phenomena and laws but systems of purposive, mostly abstract, action are its subject. Interpretation, not explanation or prediction, is the task of the system analyst.

As to the concept of "object of the real world", a naive ontological realism is the prevailing idea tightly coupled with the notion of isomorphic modelling.

• "Objects" are relative to some language. Their identification is dependent on a public reference system. How individuals may be distinguished from each other and its environment is, among other things, a matter of convention and degree.

Representations are understood as (isomorphic) mappings. It makes no sense to transfer this concept to descriptions or software reconstructions of "some sector of the real world".

• Software representations can only capture the logical aspects of conceptualizations. Their meaning and reference are not subject to morphisms but to intentional interpretation.

It is a common misunderstanding in the field of object-oriented programming that concepts are entities to be 'modelled' by classes, and that class hierarchies are ‘models' of conceptual hierarchies.

• Class hierarchies are description hierarchies based on the purely syntactic mechanisms of extension and factorization. There is no set-theoretic interpretation such as it is applied to concept hierarchies.

It is believed that real world concepts can be or at least should be formally defined rendering reasoning purely analytical and "objective".

• The inherent SE dilemma cannot be solved by formal methods. Software reconstructions are only purposive approximations, they have to be open for interpretation, hence, for change.

Last not least, SE lacks a consistent use of "model" as one of its central terms. Especially, there is confusion as to a 'model' *for* and a 'model' *of* something. There are different meanings of "model" that should not be confused.

• First, a semantic model is a formal structure satisfying a specification (model of).

• Second, a descriptive world model is a selective and purposive informal or formal description intended to capture certain aspects of the real world (model of).

• Third, a system model is a - mostly formal - prescription for a system to be implemented (model for).

• Fourth, an experimental, or explorative, model (also called "prototype") is a testbed for users (model of) or developers (model for).

What are the consequences? Epistemological misconceptions may induce severe drawbacks in research as well as in practice of SE. Especially, the assumption that only formal methods are warranting precision, validity, and objectivity is one of the most eminent mistakes. Contrarily to propagating "the cruelty of teaching computer science"[2], we need to introduce more "kindness" into SE, namely

• an analytical competence comprising social and ethnological techniques of human ("familiar") system analysis,

• a formalization discipline (mostly called "modelling", misleadingly) comprising purposive methods to assess the consequences of reduction,

• a communicative competence comprising abilities to communicate appropriate to the knowledge states and abilities of the partners, and to do teamwork,

• participation frameworks and tools for development sustaining cooperation both within development and across application domains.

The ultimate consequence may be the abandoning of SE as a self-contained discipline altogether and introducing, instead, many disciplines of SE contained in those disciplines that are directed to aspects of the "real world".

References

1. Dennet, D.: Consciousness explained. Boston 1991.
2. Dijkstra, E.: The cruelty of really teaching computer science. CACM 32 (1989).
3. Frege, G.: Funktion, Begriff, Bedeutung. Hrsg. von G. Patzig. Göttingen 1980.
4. Hertz, H.: Einleitung zu <Prinzipien der Mechanik>. In: W. Heisenberg: Das Naturbild der heutigen Physik. Reinbek 1955.
5. Keil, G.: Kritik des Naturalismus. Berlin 1993.
6. Korson, T., J. D. McGregor: Understanding object-oriented: a unifying paradigm. In: CACM 33, No. 9 (1990), S. 40-60.
7. Madsen, O. L., B. Möller-Pedersen, K. Nygaard: Object-Oriented Programming in the BETA Programming Language. Wokingham etc. 1993.
8. McDermid, J.: Introduction and overview to Part II. In: J. McDermid (Eds): Software Engineers Reference Book. Oxford 1993.
9. McDermid, J., T. Denvir: Introduction to part I, In: J. McDermid (Eds): Software Engineers Reference Book. Oxford 1993.

10. Monahan, B., R. Shaw: Model-based Specifications. In McDermid (1993)

11. Piepenburg, U., H. Züllighoven: Objektorientierte Systenentwicklung. In: W. Dzida, U, Konradt (Hrsg.): Psychologie des Softwareentwurfs. Göttingen 1995, S. 45-59.

12. Putnam, H.: Repräsentation und Realität. Frankfurt 1991 (engl. original: Representation and Understanding, MIT 1988)

13. Quine, W. V. O.: From a logical Point of View. Cambridge (Mass.) 1964.

14. Quine, W. V. O.: Zwei Dogmen des Empirismus. In: J. Sinnreich (Hrsg.): Zur Philosophie der idealen Sprache. München 1972.

15. Rumbaugh, J. et al.: Objektorientiertes Modellieren und Entwerfen. München 1993 (engl. original Object-oriented Modeling and Design, New York 1991)

16. Stokes, D.: Requirements analysis. In: McDermid (1993).

17. Strawson, P. : Individuals. London 1959.

18. Turing, A.: Über berechenbare Zahlen und eine Anwendung auf das Entscheidungsproblem. In: B. Dotzler, F. Kittler (Hrsg.): Alan Turing. Intelligence Service, 1987.

19. Wittgenstein, L.: Tractatus logico-philosophicus. Frankfurt 1963.