SACJ 32(2) December 2020
Viewpoint

Invited Lecture: Notions of 'Theory' and their Practical Consequences in the Discipline of Software 'Engineering' (including Information Systems Design)

Stefan Gruner sg@cs.up.ac.za
Department of Computer Science, University of Pretoria, South Africa

Preliminaries. In the *context of the AISSAC seminar* on information systems, to which this *invited lecture* was given, it is presumed throughout the following considerations that:

- contemporary *information systems are software systems*, whereas traditional libraries of printed books (which are indeed pre-electronic information systems) are *not* considered here;
- design and development of information systems (from capturing the initial requirements to implementation and deployment) are *software engineering* activities;
- it is therefore justified to speak about 'software engineering' (rather than about 'information systems') throughout this lecture,
- whereby the notion of 'information' is assumed to be already 'basically understood' (and is thus *not* subject to further science-philosophical definitions or critique in this contribution).

Some background literature on these preliminaries is already available (Gruner, 2016; Gruner & Kroeze, 2014). Moreover: a small 'sub set' of the considerations, which are elaborated in much detail in the following sections, had already been briefly discussed on another occasion (Gruner, 2019). My new insights and 'lessons learned' from that previous discussion, as well as those ones from the most recent AISSAC seminar itself, are of course 'integrated' into this contribution.

Gruner, S. (2020). Invited Lecture: Notions of 'Theory' and their Practical Consequences in the Discipline of Software 'Engineering' (including Information Systems Design) [Viewpoint]. *South African Computer Journal 32*(2), 293–322. https://doi.org/10.18489/sacj.v32i2.898

1 INTRODUCTION

In July 2014, the following message was publicly distributed via the ACM's SEWORLD network:

> "While evidence-based software engineering (EBSE) has attracted considerable attention from the research community, there is still a lack of interest and appreciation for the role of theory in the software engineering field. In order to make sense of all the empirical observations and evidence that researchers are gathering, we need theory that allows the abstraction of these observations into 'universal knowledge' that is useful not only to other researchers but also to software engineering practitioners. Specifically, what the SE field seems to be missing is a General Theory, such as can be found in many other academic disciplines. Examples of general theories include the 'big bang' theory and evolution theory. These 'general' theories are able to explain (or predict) phenomena within a larger context of a discipline. This is of particular interest to the empirical software engineering community, which is increasingly recognizing the importance of context of research. However, few general theories of software engineering have been proposed, and none have achieved significant recognition. In turn, software engineering remains limited to problem solving by trial-and-error and rules-of-thumb and in most cases only related to a limited area of relevance. The state of research in software engineering cannot make significant advances as new trends are emerging quickly and a systematic cumulative research tradition within software engineering has not yet been achieved".[1]

The announcement quoted above eventually lead to several publications (Johnson et al., 2015).

Topically related to the above-mentioned communication are a number of well-organized activities (Johnson et al., 2015), which include (for example) the 'GTSE' workshop series on '*General Theory of Software Engineering*'. In the GTSE edition of the year 2015, for example, 'principles of separability' were strongly emphasized, but also in this case *"one asks how to appraise the generality of these theories? And in case they are specialized sub-theories, are they amenable to combination into more general theories?"* (Exman et al., 2016). In addition to such a search for 'generality' (which is a classical, science-oriented approach to truth) it is also typical in these kind of efforts that they rarely problematize their own *notion of 'theory'* from a meta-theoretical point of view. It seems as if the meaning of the term 'theory' itself is simply taken for granted by the participants of those discourses in which no definition of the term 'theory' is given. The 'GTSE' workshop of 2015 ended with notable sentiments of frustration (Exman et al., 2016), which are at least in part also due to some intra-communal factional differences between the 'engineers' and the 'sociologists' with their different epistemological and meta-theoretical points of view (Johnson & Ekstedt, 2016, p. 182). However, nobody seems

[1]https://listserv.acm.org/scripts/wa-acmlpx.exe?A2=SEWORLD;6bdb5f2.1407 (10 July 2014).

to have asked whether such frustration was perhaps a necessary consequence of the impossibility of an attempt to (proverbially) 'square the circle' (i.e.: to solve an inherently unsolvable problem). Though, *"based upon philosophy of science and software engineering practice"*, the participants of 'GTSE 2015' already had the correct insight *"that engineering fundamentally differs from scientific disciplines"* (Exman et al., 2016), nobody seems to have consequently questioned the 'feasibility' of a science-like 'general theory' for a 'fundamentally different' engineering discipline. Numerous interesting literature references on this topic are already known (Exman et al., 2016), though not all of them can be recapitulated in this section.

A so-called 'design theory' *for* (not: 'of') software engineering was meant to be *"a theory that characterizes the elements of software problem solving in terms of the effect they have on the process of design"* (Hall & Rapanotti, 2017). Also in that publication the basic notion of 'theory' itself was not precisely defined (and was thus more-or-less taken for granted), though a distinction between 'product' theories (about the things made) and 'production' theories (about the work-steps that lead to the things made) was taken into account (Hall & Rapanotti, 2017). Very important was the assertion that *"the community"* was *"also trying to come to terms with what is meant by >>theory<<"* (Hall & Rapanotti, 2017). Everywhere in in that publication we can find references to a particular notion of 'theory' (Gregor, 2006), which, however, stems from the sub-field of *'information science'* (IS) and is thus (like the entire sub-field of IS) strongly sociology-influenced and business-management-oriented. For specific historic reasons (Kline, 2006), the IS community does typically *not* maintain the self-view of a community of engineers (Johnson & Ekstedt, 2016, p. 182; Gruner & Kroeze, 2014, p. 35). Also that publication's list of literature references (Hall & Rapanotti, 2017) contains many further interesting entries which cannot all be recapitulated in this section. Noteworthy in particular is its formalization of software-construction-related reasoning-steps in *'Gentzen style'* (Hall & Rapanotti, 2017).

In the year 2016, a *Special Section on General Theories of Software Engineering* was published by the *Journal for Information and Software Technology* (Stol et al., 2016). The most interesting publication in that special section was a paper which emphasized a *general* theory of software engineering (Johnson & Ekstedt, 2016), whilst the other papers of that special section discussed specialized theories of smaller concepts and sub-concepts. In their editorial preface, which also contains several further noteworthy literature references, the guest-editors pointed once more to *physics* as the 'reference discipline' for general theories (Stol et al., 2016), which tacitly implies that an engineering discipline (like software engineering) would (or should) ultimately have to be(come) something like a 'nature-scientific' discipline. Such an attitude, however, ignores the relative simplicity of the 'things' about which the science of physics can successfully produce 'general' theories: as soon as matters get somewhat more complicated, such as (for example) in meteorology, physics is also at loss as far as the production of substantial, non-trivial 'general' theories is concerned. Moreover: the notion of 'theory', which is expressed there (Stol et al., 2016), cannot be robustly defended science-philosophically at that point where a *'micro theory'* is defined merely as a small number of hypotheses concerning some technical properties of one particularly given software system or software *product* (Stol et al., 2016, p. 177). Such a device-specific notion of 'micro theory' is *not* consistent with the *area-*

specific notion of ‘micro theory’ in the established engineering disciplines. All in all the notion of ‘theory’ in that contribution (Stol et al., 2016) remained as vague as it had always been in the ‘GTSE’ community.

In the above-mentioned ‘general’ paper (Johnson & Ekstedt, 2016), which includes more than 170 noteworthy literature references, it is interesting see a *‘theory’ without theorems* in which various ‘models’ (graphically depicted work-flow schemes, semantic concept hierarchy graphs with notions and sub-notions, and the like) are presented as ‘theory’. This peculiar willingness to meta-theoretically accept some simple (and rather incoherent) graph models as ‘theory’ seems to be a prominent (if not even the dominant) methodological attitude in relevant communities of practitioners. Nonetheless we can find in that publication (Johnson & Ekstedt, 2016) also some more salient meta-theoretical remarks about what is ‘theory’. W.r.t. Popper as well as w.r.t. to the *Stanford Philosophical Encyclopedia* the paper stated that

> “a theory of software engineering is thus a collection of sentences, propositions, statements or beliefs, etc., about software engineering, and their logical consequences. We are aware that some definitions pose additional requirements on theories, such as being internally consistent, well-substantiated by observation, clearly delimited, formalized, falsifiable, sufficiently general, connected, or able to make sufficiently accurate predictions. Many of these are in our view, however, quality criteria rather than requirements, so that while, e.g., an internally inconsistent theory is inferior to a consistent one, ceteris paribus, the inconsistent exemplar is still, in our terminology, a theory” (Johnson & Ekstedt, 2016, pp. 182-183).

Thus: in a post-modernist spin of words, actual *non*-theories (which include more or less arbitrary ‘beliefs’ about software engineering) have been elevated in to the noble status of ‘theories’ (Johnson & Ekstedt, 2016) simply by decree, namely the decree that the hitherto indispensable criteria of theory-*ness* (Bunge, 1998a) shall henceforth be merely regarded as some merely accidental ‘quality’ attributes. As a consequence of such a post-modernist meta-theoretical attitude concerning the notion of ‘theory’, anybody can now come up with more or less arbitrary ‘theories’ of software engineering’: this must ultimately lead, indeed, to the intellectual frustration which the organizers of the above-mentioned GTSE 2015 workshop have noticed (Exman et al., 2016). To be fair, it must also be appreciated that the above-mentioned authors consider *“the ability to make predictions about the empirical world”* as a meta-theoretical quality criterion *“of particular importance”* (Johnson & Ekstedt, 2016, p. 183). Nonetheless the question remains un-answered: what exactly about software engineering (as a whole) shall be ‘predicted’?

Therefore, in this lecture I raise and defend the *conjecture* that the above-mentioned call for a ‘general theory’ of software engineering is ‘wrong’ in the sense that *inappropriate notions of ‘theory’ will mislead the discipline of software engineering into an intellectual cul-de-sac* and will potentially distract the software engineering practitioners from successfully applying the many useful ‘micro theories’ (Stol et al., 2016) which are already available in several specialized sub-areas of software engineering (including information systems design). I conjecture that the recent quests for a 'general theory' in software engineering are so-to-say *'mythical'* in their desire and flawed particularly in all those instances in which the *specific differences between science and engineering* are not seriously taken into account, whereby this critique includes a number of papers in which merely some superficial 'lip service' is paid to this difference. In this context we must also not forget the historic vulnerability of the software engineering discipline as far as the lore and the lure of various *'myths'* is concerned (Brooks, 1995). In particular the *'myths' of software engineering's 'scientification'* have recently come under the scrutiny of professional historians of technology (Hellige, 2008).

The practical consequence of those futile pursuits are not merely the already mentioned intra-disciplinary 'frustration', but also an unfortunate distraction of precious 'intellectual resources' from many noteworthy software engineering problems (both technical and philosophical) to which satisfactory and 'systematic' solutions have not yet been found. Thereby, the 'highest level' of 'solution' that can possibly be achieved in the realm of software engineering is automation: a given problem in software engineering is considered 'fully understood' when it can eventually be solved repeatedly by algorithmic means. All other problems in software engineering can maximally be considered 'partly understood', (which is of course still much better than not understood at all). Insight into the fundamental insolubility of some problems by any means (for example the Halting Problem or the 'Perpetuum Mobile') belongs to the realm of science: problems that are in principle (absolutely) unsolvable (and scientifically confirmed as such) are therefore methodologically excluded from the domain of (solution-producing) engineering. For illustration: *if* I had 'fully' understood how Requirements Engineering (within any SE project) ought to be done, then I should be able to implement a Turing Machine that would be able to find industrial 'employment' as a Requirements Engineer. This example shows immediately that Requirements Engineering, as a whole, is far away from being 'fully understood' (in the above-mentioned technological sense of the term 'full understanding'), such that any 'general theory' of Requirements Engineering, which would explain the blue-print and predict the run-time behavior of such a (fictive) Turing Machine, is nowhere to be seen.

To compose the argument of my lecture as a whole,

- I recapitulate the *basic notions of 'theory'* from the standard literature in *philosophy of science*;
- I recapitulate some of the most fundamental *differences between science and engineering*;
- I recapitulate the historic fact that the *engineering* disciplines are based on the successful development and application of domain- and sub-domain-specific *'micro theories'* which are never general;
- I argue that applicable 'micro theories' already exist for several domains and sub-domains in the discipline of software engineering which includes the sub-domain of information systems; (their existence shall of course not prevent us from finding more and better ones as time goes by).

Subsequently, what is needed most (in addition to the continuing improvements of those 'micro theories' themselves) is that a large majority of software practitioners will eventually pick up and actually apply those 'micro theories', which are already available for practical use, instead of 're-inventing the wheel' ad-hoc for each and every new software development project.

As far as the frequently *perceived 'difficulty'* of such theories (from the perspective of non-academic practitioners) is concerned, we will eventually have to find the *professional courage* to declare together with Edsger Dijkstra (1975):

> "Don't blame me for the fact that competent programming, as I view it as an intellectual possibility, will be too difficult for 'the average programmer': you must not fall into the trap of rejecting a surgical technique because it is beyond the capabilities of the barber in his shop around the corner".

In fact Dijkstra was not the first one to have recognized some characteristic similarities in the theoretical world-views of surgeons and engineers (Bamm, 1956, p. 154).

The envisaged practical applications of the already existing (albeit difficult) 'micro theories', however, can be successfully achieved only by further specialization of software 'engineering' into more and finer-structured sub-disciplines in the same manner in which any general *'hardware engineering' does not exist* apart from its various sub-disciplines (such as railway engineering, mechanical engineering, electrical engineering, chemical engineering, aeronautical engineering, naval engineering, mining engineering, and so on). They all possess their own domain-specific 'micro theories' which prevent them from the kind of *pre-theoretical 'crafting'* by which much of software construction is still characterized to-date. For the topic of 'pre-theoretical crafting versus science-based engineering' another warning by Edsger Dijkstra (2000) is relevant and will most likely remain relevant for many years to come:

> "The required techniques of effective reasoning are pretty formal, but as long as programming is done by people that don't master them, the software crisis will remain with us and will be considered an incurable disease. And you know what incurable diseases do: they invite quacks and charlatans in, who, in this case, take the form of software engineering gurus".

2 WHAT IS 'THEORY'?

In his several hundred pages strong book on a topic of social and political ethics, the philosopher Margalit frankly admitted to have *"no theory"* on this matter (which is highly unusual for public intellectuals and men of letters who typically identify themselves as 'theorists') and continued to explicate meta-theoretically several notions of 'theory' by Hilbert, Gödel, Carnap, and Reichenbach (Margalit, 1996). Accepting their definitions of the notion of 'theory', he concluded that his own socio-political and philosophical elaborations were *not* 'theoretical' in the strictest sense of the term (Margalit, 1996).

Also in our case of software engineering it is not adequate to demand the elaboration of a 'theory' or 'theories' without any meta-theoretical considerations concerning the notion of 'theory' as such. However, the science-philosophical and meta-theoretical literature on the topic of 'theory' is so vast and detailed that it cannot be adequately recapitulated in this section. Only a few of its most important aspects will be cursorily summarized in the following paragraphs. For a thorough introduction into the science-philosophical meta-theory of scientific theories the reader is referred to the usual literature (Bunge, 1998a).

The original meaning of the *classical Greek* word $\vartheta\epsilon\omega\varrho\iota\alpha$ was 'speculation' or 'contemplation', whereby a philosopher 'gazed' at the objects of interest with the 'inner eye' of the mind in search for truth. This type of 'gazing' is still considered as 'essential' in Husserl'ian types of 'phenomenological' philosophy. According to Aristotle the term $\vartheta\epsilon\omega\varrho\iota\alpha$ can also be understood as the 'thinking about thinking' (i.e.: the philosophical self-reflection of thought) in opposition to 'practice' or 'praxis' (Hoffmeister, 1955).[2] This notion of 'theory' was closely related to the notion of 'pure reason' or 'pure knowledge' in its systematic order, *regardless of any utilization* for specific purposes (Hoffmeister, 1955). From an etymological point of view, the word $\vartheta\epsilon\omega\varrho\iota\alpha$ is also related to the Greek word $\vartheta\epsilon o\varsigma$ (God) and implies thus some connotation of 'divinity' in the act of theoretical philosophical speculation (Hoffmeister, 1955). In the *newer philosophy*, 'theory' is opposed to 'experience' (not to 'praxis').[3] Theoretical knowledge is in this context the knowledge obtained through thinking, the scientific explanation of particular phenomena on the basis of fundamental principles, the conglomeration of many specific insights with the help of case-independent laws, and the like. Thereby a theory distinguishes itself from one individual 'law' or one individual 'hypothesis' (which also 'captures' a particular fact or phenomenon) by its far more comprehensive character. Therefore, every *science* eventually aims at the development of a theory in this modern sense of the term which is always understood as *providing overview through description and insight through explanation* (Hoffmeister, 1955). Moreover, a scientific theory is not only more comprehensive than an individual law: it must also be possible to *logically deduce* particular individual laws from the totality of the theory with which these individual laws are associated (Bochenski, 1954, 1965).

An apparently paradoxical attitude was held by the philosophers of *phenomenology:* the ideal of philosophical phenomenology, specifically Husserl's, was *pure beholding* under *exclusion of all theory and all tradition* (Bochenski, 1954, 1965). This philosophical programme implies that the classical Greek notion of $\vartheta\epsilon\omega\varrho\iota\alpha$, which had been identified with the act and attitude of contemplative beholding, was no longer the phenomenologist philosophers' notion of 'theory' at the beginning of the 20th century:

> "A peculiarity of 20th century science is that the most important scientific activity—the deepest and most fertile—is centered around theories rather than around stray questions, data, classifications, or stray conjectures. Problems are posed and data

[2]This conceptual difference does not prohibit thinking about the essence of doing, i.e.: 'theory *of* praxis'.

[3]This conceptual difference does not prohibit thinking about the essence of experience, i.e.: 'theory *of* experience'.

> are gathered in the light of theories and with the hope of conceiving new hypotheses that may in turn be expanded or synthesized into theories; observations, measurements and experiments are executed not only to collect information and generate hypotheses but also to test theories and find their domain of truth; and action itself, to the extent to which it is deliberate, relies more and more on theories—for the better or for the worse. In short, an emphasis on system—on empirically testable theory, of course—rather than on raw experience is what characterizes contemporary science" (Bunge, 1998a).

Moreover, the notion of 'theory' must neither be defined so narrowly that only mathematics and physics may be (exclusively) regarded as possessing genuine theories (which would be the position of extreme 'physicalism') nor so widely that any arbitrary kind of opinion about anything may be called a 'theory' (Charpa, 1996). Thus, theories can express *scientific* opinions *only if* they fulfill particular requirements and functions with regard to data, principles, and their own internal *logical structure* (Charpa, 1996). As far as the relations between theories and data are concerned, three functions are of highest importance, namely: explanation, description, and prediction. Additionally also the generation as well as the justification of data can belong to the functions of theories (Charpa, 1996). Moreover, theories can also appear as concretizations of metaphysical systems or as application cases of methodological principles. For example: the physical theories constructed by Kepler or Newton were embedded into far-reaching nature-philosophical speculations which consider the 'essence' of 'nature' and which include even theological thoughts about divine manifestations in the physical realm. Even in the 'secularized' 20th century, Albert Einstein structured his physical theories according to his own meta-theoretical maxims and meta-physical speculations (Charpa, 1996).

Such as the *notion* of 'theory' must neither be too narrowly nor to widely defined, also the *semantic extent* of a theory must neither be too narrow nor too wide (Charpa, 1996). Overly general theories would tell us (so-to-say) 'nothing about everything', whereas overly specific theories would tell us (so-to-say) 'everything about nothing'. Either way, theories can be regarded as *systems of propositions* which can even be represented formally in an axiomatic fashion (Charpa, 1996). The 'theorems' of such a system are of law-like type in such a manner that formal-logical deductions (in the formalism of the theory) and material explanations of observable phenomena (about which the theory speaks) are closely related to each other (Charpa, 1996).

The above-mentioned notion of 'theory' is closely related to the notion held in *mathematical logic* (Schöning, 1992), according to which a set of formulæ $\mathcal{T} = \{F_1, F_2, \ldots, F_n\}$ is a 'theory' *if and only if* $\forall F_i \in \mathcal{T}\ \exists \mathcal{F} \subseteq (\mathcal{T} \backslash F_i)$: $\mathcal{F} \models F_i$, as well as $\forall \mathcal{F} \subseteq \mathcal{T}\ \exists F \in \mathcal{T}$: $\mathcal{F} \models F$. Two sub-types of these types of formalized theories, are the *Hilbert-Carnap-Reichenbach* type (HCR) as well as the *Gödel-Chomsky* type (GC) (Margalit, 1996):

- In the HCR notion of 'theory' we assume to have intuitive and immediate access to a domain D of 'primitive' entities or basic facts (a.k.a. axioms) on top of which the theorems of a formalized theory are 'formalistically' constructed in such a manner that further 'information' can be 'obtained' via the formalized theory's rules of deduction (Margalit, 1996).

- In the GC notion of 'theory', on the other hand, it is presumed a-priori that a domain and a co-domain of entities already exist independently of each other (i.e.: the theorems of a theory on the one hand, and all theory-independent 'truths' on the other hand), such that 'theorizing' becomes a matter of how to find suitable and consistent mappings (typically partial ones) between the domain and the co-domain. This type of 'theorizing' can possibly be bi-directional, whereby not only the formal theory can be used to deduce and reveal hitherto 'hidden' information, but also (vice versa) the 'quality' and 'suitability' of the theory can be judged on the basis of our intuitive insights into the pre-existing 'truths' which constitute the theory's co-domain (Margalit, 1996).

For example: Gödel and Tarski have famously demonstrated that no formalized arithmetic theory $\mathcal{A}$ can 'detect' all 'truths' within the same arithmetic system and that no formal language $\mathcal{L}$ exists to consistently define the notion of 'truth' within the same language $\mathcal{L}$ (without resorting to some higher-level meta-language). Both results are closely related with each other from a semantic point of view.

However, more common than those purely formalistic notions of 'theory' are nowadays the so-called *'model-oriented'* notions of 'theory' (Charpa, 1996). According to the meta-theory of model-oriented theories, their typical functions (w.r.t. data, etc.) are carried by 'models' or 'structures' rather than by linguistic-logical expressions. Thereby the underlying notion of 'model' can differ strongly from discipline to discipline. In pure logics (for example) a 'model' of a given formula $\mathcal{F}$ is a set of object-values which (by way of substitution or 'interpretation') can make $\mathcal{F}$ 'true'. This notion of 'model' differs strongly from (for example) a globe as a 'model' of planet Earth on the desk of a geographer (Freudenthal, 1961). Within this model-oriented school of meta-theory, Suppe's and van Fraassen's *semantic* approach can be distinguished from Sneed's and Stegmüller's *structuralist* approach. In these schools, 'explanations' as well as 'predictions' are *assertions about relations* between the theoretical models and those phenomena to which those models are 'applicable'. In this science-philosophical meta-theory of 'modellism', 'corroboration' and 'refutation' of hypotheses are no judgments about the 'goodness' of a theory as such: rather they refer to a theory's *limits of applicability* as far as the permissible model-reality-relations are concerned (Charpa, 1996). Hence these model-oriented meta-theories can be broadly classified as 'instrumentalist', 'conventionalist', or 'operationalist' meta-theories: from these points of view, theories merely have to 'work well' (or 'be useful') to some lesser or greater extent.

To understand the differences between 'model-oriented' theories and the above-mentioned theories of type HCR it is important to remember that the leading members of the *Circle of Vienna* presupposed the existence of *basic sentences* (Protokollsätze) along the lines of Wittgenstein's *Tractatus Logico-Philosophicus* which were presumed to correspond immediately with basic 'facts' of the objective physical reality. Model-oriented theories, by contrast, 'speak' directly only about their models which occupy an intermediate position 'between' theory and the external physical reality.

In any case, theories serve the purpose of summarizing, coordinating, reproducing, explaining and predicting phenomena (Götschl, 1980). Theories thus comprise entire classes of phenomena in such a manner that they can be regarded as scientific knowledge or scientific insight (Erkenntnis). In this sense theories are primarily conceptual auxiliary-means which enable the *separation of non-scientific from scientific knowledge* and which bring the latter one into an inter-subjectively objectified form. Thereby, theories are more than merely additive agglomerations of data, observations, or empirical laws: from theories it is possible to deduce predictions of future phenomena which had not been explicitly taken into account when the theory had been constructed. Especially the theories of modern science allow not only for the deduction of individual predictions or particular laws, but also for the deduction of entire sub-theories (more special theories) from even 'more general' (super) theories. For the classification of scientific theories as 'empirical' it must be possible to check them by procedures of experience (such as, for example, measurement). To be inter-subjectively acceptable, these procedures must comply with well-defined norms and methods without which it would not be possible to distinguish between scientific and non-scientific types of knowledge. Moreover: theories may have different types and functions in different scientific disciplines, whereby also their accepted empirical checking procedures may vary from discipline to discipline (Götschl, 1980, pp. 636-637):

- For *mathematical* theories only their intrinsic logical consistency is necessary, though (according to Hilbert) a number of additional meta-theoretical requirements can be stipulated such as (for example) completeness or the mutual logical independence of its axioms.

- Consistent *empirical* theories must augment those formal-logical properties with experiential 'contents'. Only when its 'semantically empty' formal calculus is appropriately interpreted, a 'mathematically written' theory can be understood as an 'empirical' theory (Götschl, 1980, p. 637). Theoretical physics provides the best-known examples of theories of this type.

Because a theory is 'carried' by concepts (notions) and their relations, it must be clarified which types of notions are admissible for the construction of theories in such a manner that the theories themselves can be used for deducing testable predictions of future phenomena. Before theories are quantified (such as in physics) one can usually find *qualitative theories* for which it is hardly possible to assess the 'weight' or the 'relevance' of their various constituting elements. Nonetheless, also the theoretical notions and variables of qualitative theories must meet some basic testability and explain-ability requirements, without which it would not be possible to assert that a given theory is applicable to a particular domain (Götschl, 1980, p. 638). Because theories can only approximate 'the truth' partially (i.e.: never reach 'the full truth' in its totality) it is an important meta-theoretical requirement that *successively or concurrently competing* theories ought to be *comparable* to each other: the conceptual and logical pre-conditions of comparability became thus a central meta-theoretical problem in modern philosophy of science (Götschl, 1980, p. 642). In Sneed's structuralist meta-theory of theories, a scientific theory appears as an ordered pair $T = (K, I)$ for 'Kernel' and 'Intended domain of application' in such a manner that K must appear as invariant in all of T's applications. Moreover, according to Sneed, K must be sufficiently general such that different (though not completely independent) domains of application for T can be found. Only by applying a theoretical structure K to one of its intended domains (e.g.: particular physical systems) the theory's empirical assertions emerge. In other words: the scientific assertions (hypotheses) generated with help of a theory must be strictly distinguished from their underlying theory itself. This implies that the elements of I appear as 'models' of K in a formal-logical model-theoretical sense: thus they *constitute* those realms of entities about which a given theory can truthfully 'speak' (Götschl, 1980, p. 644). On such a basis (as shown by Stegmüller) it also becomes possible to meaningfully compare two successively or concurrently competing theories against each other (Götschl, 1980, pp. 645-646).

From a science-philosophical point of view there is thus a tight connection between various notions of 'theory' and various notions of 'model', though these conceptual connections (as well as the various notions of 'model' themselves) cannot be further discussed within the limited scope of this section. A more comprehensive critique of Stegmüller's meta-theory can be found in the science-philosophical literature (Hübner, 1978, 1983).

From a somewhat less 'subtle' point of view, *three contemporary notions of 'theory'* can be distinguished (Seiffert, 1992):

1. A very general notion of 'theory' comprises everything which is not immediate action (or praxis) and which can also be applied to the non-scientific realm (for example when a football team is 'theoretically' discussing the tactics before a tournament).

2. A slightly more specific notion of 'theory' refers to various kinds of intellectual or scholarly or academic doctrines in various disciplines, particularly in the *humanities and social sciences* (for example a 'theory of education' in the pedagogical faculty, the 'Critical Theory' propagated by the socio-philosophical School of Frankfurt, or the 'theory of science' as synonym of 'philosophy of science').[4]

3. A very specific notion of 'theory', as it was propagated by the science-philosophical *Circle of Vienna*, refers to scientifically solid knowledge as a result of a methodologically sound combination of rational thought and empirical observations (for example the theory of gravitation in classical Newton'ian physics).

Obviously the *extension* sets (E) of those three notions include each other: $E(1) \supset E(2) \supset E(3)$.

As far as this lecture's domain of interest (namely the techno-philosophy of software engineering) is concerned, it seems that (2) is the notion which the senders of the above-mentioned SEWORLD communication might possibly have had in mind when they called for a 'general

[4]'Intellectual' theories especially of this type can get into the dangers of ideologization (Greeley, 1975).

theory' of software engineering. This vague notion of 'theory' is typically found amongst public intellectuals, political ideologists, literary essayists, men of letters, and the like, for whom 'theory' is by-and-large anything what they think or write about in their capacity as public intellectuals (Greeley, 1975; Schlette, 1975). The above-mentioned counter-example notwithstanding (Margalit, 1996), the popular *Theory of Everything* (Wilber, 2000) may be mentioned as an illustrative example.

A similarly vague ideological notion of 'theory' can also be found in some 'counter-cultural' doctrines concerning the discipline of informatics, which includes software engineering, by-and-large (Coy, 1992). As an informatician strongly influenced by the School of Frankfurt's 'Critical Theory', Coy doubted the intra-disciplinary coherence of informatics. Thus he classified informatics as akin to the social sciences, and consequently demanded the elaboration and production of a so-called *theory of informatics* in general (Coy, 1992). His computer-philosophical work was explicitly aimed at separating the discipline of informatics (including software engineering) from its formal-logical traditions (Coy, 1993) as well as at pushing this discipline into the realms of some socio-culturalist postmodernism (Coy, 1992). According to his call for a to-be-construed 'theory of informatics', individual and particular ad-hoc approaches to a 'theory of design' in the sub-fields software design and hardware design have remained separate and disparate to-date (Coy, 1992). His ideas were thus quite consistent with what the senders of the above-mentioned SEWORLD communication have stated in their call for a 'general theory' of software engineering. From such 'deficits' (according to Coy) 'follows' the (alleged) 'necessity' to develop a 'theory of informatics', the purposes of which ought to be:

- to *describe* the concepts, methods and potential applications of informatics by-and-large;
- to *determine* the discipline's status in comparison to the other academic and technical disciplines from a science-philosophical point of view (Coy, 1992).

Such a general 'theory of informatics' was also intended to reveal the social and political implications of information-technical computer applications in the cultural realm (Coy, 1992). The affinity of such a concept of 'theory of informatics' with the School of Frankfurt's concept of 'Critical Theory' is obvious. However, because of its practical fruitlessness, Coy's ideology eventually 'fizzled out' (Hellige, 2012): except of a few neo-marxists in some German faculties of computing hardly anybody was interested in developing any such 'theory', and I predict the same fate for similar political 'theories of informatics' from the English-speaking parts of the world (Galloway, 2004).

For the sake of completeness I also briefly mention a very peculiar notion of 'theory' that had been suggested in our discipline by the well-known informatician and software engineer Peter Naur with his claim that the act of programming (or software construction) would be 'theory-building' (Heidelberger, 1993; Naur, 1985). This is, however, not the case. Whatever it might be that is being 'built' by (or through) programming (or software construction): it is certainly not a theory in any of the science-philosophically endorsed meanings of the term ‘theory’. Surely we can gain interesting insights and professional experience by way of software construction (like a medical practitioner will gain additional insights and professional experience every day in the clinic); however those personal insights must still be properly ‘objectified’ before they can be included into the ‘corpora’ of teach-able theories. To claim that a piece of software would be a ‘theory’ (*of what?*) is merely a mis-use of the word >>theory<<, though it is indeed possible to behold a piece of software as a *model* of some real-world phenomenon which can be computer-simulated by ‘running’ such software on the machine (Winsberg, 2010).

After the literature discussions of above it should have become clear that the various ‘general software engineering theories’, which were proposed by various members of the GTSE community, are far away from meeting the above-mentioned meta-theoretical criteria of theory-*ness*. Indeed, the central conjecture of this lecture is my claim that this gap will never be closed (as a matter of principle) *unless* the underlying notion of ‘theory’ is post-modernistically trivialized. By analogy: there also exists no ‘general theory of medicine’, though the academic hospitals with all their various specialized clinics, departments and their area-specific ‘partial’ theories of the human body are functioning very well in their daily practice (Silva et al., 2018). More positively: if a philosopher like Margalit is not afraid of admitting to have *“no theory”* of the modern society (1996), then also software engineers need not be afraid of the absence of a ‘general theory’ of software engineering. Well-defined and area-specific ‘micro theories’ in combination with our discipline’s many decades of professional experience will suffice. In other words: what we need at this point in the history of software engineering is not so much an over-arching meta ‘theory’ about the entire discipline of software engineering as an abstractum. Instead, what we need is an array of ever better and finer object-theories within the discipline (about concrete, well-defined and ‘manageable’ intra-disciplinary phenomena and sub-phenomena) which can enable the practitioners of software development to achieve their own transitions from technicians (artisans and craftsmen) to genuine engineers (if they are only willing to apply those intra-disciplinary object-theories in their specific domains and sub-domains of expertise). By analogy to older engineering disciplines, as well as the science-based practical discipline of medicine (Silva et al., 2018), the elaboration of such specialized, intra-disciplinary, small-scale software engineering micro theories is one of the main purposes of informatics as a both empirical and formal science in communication with the technical practice of engineering (Broy & Schmidt, 1999). Hence: any proposed ‘general theory’ of software engineering can be

- *neither* of Seiffert’s general type 1 (which would be un- or anti-scientific);
- *nor* of Seiffert’s philosophical type 2 (which would be counter-productive as far as the yet-to-be-achieved transformation of software ‘engineering’ into an array of genuine and highly specialized engineering disciplines is concerned);
- *nor* of Seiffert’s scientific type 3 (because the entire business of software ‘engineering’, as a whole, is not a sufficiently ‘simple’, well-defined and human-independent natural phe-

nomenon which could be objectively described with help of an axiomatized formal system in such a manner that law-based Hempel-Oppenheim explanations (Bunge, 1998b) could be provided about this phenomenon by-and-large).

As far as the sub-area of information systems (within the domain software engineering) is concerned, similar science-philosophical considerations can be made about its currently dominant meta-theories (Gregor, 2006; Oates, 2006, pp. 293-296).

3 SIMPLIFIED EXAMPLE: WHAT ENGINEERS ‘ADD’ TO SCIENCE

In the following paragraphs I methodologically distinguish between ‘scientist’ and ‘engineer’ as different roles (notwithstanding the possibility for one sufficiently competent expert to play both roles in personal union, albeit not at the same point in time).

Imagine how a scientific chemist in the laboratory synthesizes some substance S in a small laboratory glass on the heat of a Bunsen flame from a mixture of more basic substances, $S_1, \ldots, S_n$. Having thus demonstrated the existence of S, the chemical scientist is now determined to explain *why* S emerged from the basic ingredients $S_1, \ldots, S_n$ under the influence of heat. After some scientific explanation has been (hypothetically) found, the chemist’s intellectual task (w.r.t. S) is by-and-larged accomplished (from a purely scientific point of view), though the individual hypothesis concerning S will still have to be ‘integrated’ into a more comprehensive scientific theory of chemistry.

Now assume that (somewhat later) an *applied* chemist has found out that the substance S *can be used* for a number of interesting practical purposes. We could think (for example) about chemicals which are applied in pharmacy, or chemicals which are applied to wash out the the gold-containing ores in the mining industry. Thus there arises a social (commercial and/or political) *desire to have* substance S available as a ‘commodity’ in sufficiently large quantities. Like the ‘theoretical’ chemist, the ‘applied’ chemist is still interested in finding scientific answers to ‘why’-questions, however with possible applications already in mind: for example, the applied chemist might want to find out scientifically why (for what ‘deeper’ reasons) the new substance S ‘works’ so well for the purpose of washing out the ores in the gold mine.

At this point the chemical *engineer* enters the scene (Auyang, 2003). Knowing already that S can be synthesized in principle (as demonstrated by the chemical scientist) and that S can be practically utilized (as shown by the applied chemist), the chemical engineer’s task is now to find an innovative method for the industrial mass-fabricating of S, as cheaply (economically) as possible, in very large quantities. This problem is outside the scope of interest of the purely scientific chemist in the laboratory, whereas the purely scientific question (why was it possible to synthesize S from $S_1, \ldots, S_n$ in the first place?) is outside the professional scope of interest of the chemical engineer (Auyang, 2003). Whilst the scientific chemist in the laboratory had it fairly ‘easy’ to produce a very small quantity of S with help of rather small tools at hand, the ‘up-scale’ to the economically frugal mass-production of S confronts the chemical engineer with a large number of additional *technical* problems which are all outside the scope of chemistry as a pure natural science:

- Whereas S and $S_1, \ldots, S_n$ may be quite harmless in their tiny laboratory-appropriate quantities, large accumulations of S and $S_1, \ldots, S_n$ are likely to be dangerous (Auyang, 2003), such that safety precautions must be considered.
- Whereas in the small laboratory glass on the intense heat of the Bunsen flame the chemical reaction of $S_1, \ldots, S_n$ to S happens quasi-immediately and quasi-everywhere (homogeneously) in the glass, this is not so in a large industrial vessel into which enormous quantities of $S_1, \ldots, S_n$ are pumped through pipelines: in such an 'up-scaled' setting the chemical engineer must also think about
  - how to bring the heat energy from the walls of the huge industrial vessel quickly into the center of the vessel, such that the chemical reaction does not merely occur in close proximity to the walls of the vessel (Auyang, 2003), and
  - how to mechanically mix or stir the in-pouring currents of $S_1, \ldots, S_n$ fast enough such that the total size of the 'reactive surface' (or 'interface') between the chemically reacting substances becomes as large as possible inside the vessel (Auyang, 2003).
- Whereas the small laboratory-quantities of $S_1, \ldots, S_n$ as well as the gas for the Bunsen-flame are rather *cheap* from a financial point of view, such that economic frugality is not a genuine scientific concern, the day-to-day operation of a large chemical factory for the mass-production of S is very expensive, such that frugality considerations are highly important from the chemical engineer's point of view (Auyang, 2003).

All this, which the purely scientific chemist in the laboratory has the luxury to ignore, must be theoretically and empirically known by the chemical engineer. Table 1 (in which the 'applied' chemist 'between' the 'pure' chemist and the chemical engineer is not shown) summarizes these epistemological differences to illustrate that *no* 'general theory of chemical engineering' could possibly cover *all* of those matters (including the scientific chemical laws of $S_1, \ldots, S_n$ and S, all the problems of mixing and stirring a chemical 'soup' in a large vessel, all the problems of heat transport from the walls to the center of the vessel, all the necessary safety considerations, all the economic frugality considerations, and so on): notice particularly the difference between 'effective synthesis' and 'efficient industrial fabrication'. In other words, *"engineering knowledge is so vast and diversified as to defy a strictly unified treatment in any event"* (Vincenti, 1990, p. 13).

In this scenario, moreover, an all-comprising 'multi-theory' of so many different and divergent conceptual details could not honestly be called a coherent '*general* theory of chemical engineering', all the theorems of which would have to be strongly 'connected' with each other (Bunge, 1998a). *"Device-specific theories that depend on special approximations are a case in point"* (Vincenti, 1990, p. 14). To answer the 'why-exists' question about S, the scientific chemist

Table 1: Example: Different Professional Interests in Chemistry

Professional Knowledge-Interest	Scientific Chemist	Chemical Engineer
Which kind of S is *impossible*?	$\surd$	—
Why can S at all *exist*?	$\surd$	—
How can S be *effectively synthesized*?	$\surd$	$\surd$
How can S be *efficiently fabricated*?	—	$\surd$

must resort to ontologically 'deeper' knowledge at the level of physics (molecules, atoms, protons, neutrons, electrons, mutually exclusive quantum-states, and so on) which the chemical engineer does not need to have in order to be able to successfully carry out the required engineering tasks. At the ontologically 'deeper' level of the 'most basic' natural entities the scientific chemist (however not the chemical engineer) resorts to a 'general theory' which is relevant in and for the chemist's natural science.

4 META-THEORETICALLY RELEVANT DIFFERENCES BETWEEN SCIENCE AND ENGINEERING

After the motivating example of above, the subsequent paragraphs recapitulate some meta-theoretically salient *differences between science and engineering* from a more 'philosophical' point of view. Relevant literature on this topic is widely available (Aravena-Reyes, 2018; Bristol, 2018; Pirtle et al., 2018; Schiaffonati, 2018; Wang & Li, 2018).

Hints at the 'big bang' theory (astrophysics) or the Darwinian theory of evolution (biology), as they were made by the SEWORLD communication quoted above, are inappropriate and misleading in a context of software construction for mainly one reason: both astrophysics and biology are natural sciences, whereas software 'engineering' is not. Whereas science is concerned about *what is*, engineering is concerned about what *ought to be* (what shall be done and how). However, a 'general theory' about what ought to be is nowhere to be seen. Nonetheless, in our current historic era, science and engineering (or 'technology') are often confused and conflated with each other (Bunge, 1998b, p. 406). For this reason it is important to remind ourselves of a number of crucial differences between science and engineering. These differences are all relevant for the 'possibility'-question concerning any 'general theory' of software engineering, too. First of all, 'technology' typically refers to

- material *artifacts* (objects, devices, processes), as well as the knowledge about them;
- and the *activities* aimed at the satisfaction of specific human or social needs through the devising of appropriate artifacts, as well as the knowledge about how to conduct such activities (Arageorgis & Baltas, 1989).

Whereas science aims at increasing and rationalizing knowledge by establishing better theories, technology aims at satisfying social needs (Arageorgis & Baltas, 1989, p. 212). Nevertheless there are a number of research-oriented disciplines (including informatics) in which both scientific and technological concerns interfere. This observation implies that the difference between science and technology cannot be merely a matter of different goals: epistemological and methodological issues also play their roles in this constellation (Arageorgis & Baltas, 1989, pp. 212-213). In contrast to traditional (pre-scientific) 'craft technologies', which still occur within software engineering too, solutions of technical problems based on scientific theories lead to '*scientifically attested* technologies' (Arageorgis & Baltas, 1989, pp. 213-214). Therein, *theoretical models* (which are based on relevant parts of the applicable sciences) bridge the gap between what a scientific theory represents and the state of the world which a particular technical enterprise tries to produce. Nevertheless there are still many technical processes or entities which cannot be modeled in such a manner (Arageorgis & Baltas, 1989, p. 214; Vincenti, 1990). Thus, a peculiar *mixture* of pre-scientific crafts and scientifically attested techniques is *essential* in all 'technology', as well as also in the practices of scientifically supported medical healing (Bamm, 1956, chpt. 5), in spite of any epistemic progress in the underlying auxiliary sciences (Arageorgis & Baltas, 1989, p. 215): for comparison consider the well-known notion of *tacit knowledge* (Polanyi, 2009) and those forms of engineering 'intuition' which stubbornly resist the 'codification' and formalization (Young, 2018) that would be needed for the formulation of solid theories.

A formal-logical analysis of this gap reveals differences between *nomological* statements (scientific), *nomopragmatic* statements (technological) and *engineering rules* (technical): their logical implication relation is only uni-directional, not bi-directional (Bunge, 1998b, pp. 149-150). A material (non-formal) reason for this difference can be provided as follows: The technical production of complex artifacts involve 'reality' in a multitude of aspects. Not only entities belonging to the domains of different sciences, but also considerations about desirable properties like feasibility or reliability are always involved (Vincenti, 1990). Because each science can deal with only the one particular aspect of reality, which is 'idiosyncratic' to it, a technological problem resides outside the scope of any single science (Arageorgis & Baltas, 1989, p. 219) and, hence, outside the reach of any 'general theory' if 'general theory' shall be understood as *scientific* theory.

Though the different aspects of the real phenomena involved in a technical problem are 'incoherent' in the sense that each one belongs to the topical domain of a different science, the different sciences can well cooperate towards the solutions of technical problems in inter-disciplinary relations (Arageorgis & Baltas, 1989, p. 221). These inter-scientific relations are not formed ad-hoc for each and every particular technological problem at hand. Instead, the resemblances and analogies, which characterize various technical problems, lead to suitable allocations of such problems into larger 'families' and thus to the establishment of more or less permanent groups of sciences around those families of technical problems (Arageorgis & Baltas, 1989, p. 222). More recently those families of problems have been called '*Mode-2 objects*', which are either devices or research goals that have the potential to 'crystallize' those inter-disciplinary transactions (Nowotny et al., 2001, p. 144). In the realm of informatics, for example, there are a number of technical development problems which require the joint involvement of artificial intelligence, robotics, linguistics, psychology, etc.

The various specialties of engineering (chemical, electrical, and the like, however *never*: unspecific 'hardware engineering') are thus inter-disciplinary branches which became institutionalized via such processes of problem-induced 'hybridization'. In other words: one can characterize the various engineering disciplines as problem-based specialties whereby the various communities of engineers are primarily identified by their central concern in solving more or less precisely defined *types* of technological problems (Arageorgis & Baltas, 1989, p. 222), whereby types of '*devices*' (like ships, cars, or bridges) correspond to those types of technological problems (Vincenti, 1990). Thus the 'curriculum' of a 'school' of engineering accurately reflects the heterogeneous basis of the engineering disciplines: Besides the genuinely scientific courses and textbooks one can also find in such schools several courses which correspond to the so-called *theories of engineering* which are, however, *not* 'general'. They can be divided into *factual* theories and *operational* theories (Arageorgis & Baltas, 1989, p. 222; Bunge, 1998b, chpt. 11; Vincenti, 1990), or, in somewhat simpler terms: theories which provide '*product* knowledge' versus theories which provide '*process* knowledge' (Auyang, 2003).

Thereby, factual theories of engineering deal with the various phenomena and effects involved in the structure and the function of technical artifacts. They also include suitable methods for analyzing and controlling those phenomena and effects. All in all, the factual theories of engineering are needed for understanding and explaining such phenomena and effects on the basis of their underlying scientific theories as far as they are practically applicable. However, because of the high degree of complexity of the technical phenomena they deal with, the factual theories of engineering must often remain at a rather superficial, phenomenologically descriptive level (Arageorgis & Baltas, 1989, p. 222; Bunge, 1998b, chpt. 11; Vincenti, 1990). Chemical engineering has been discussed as one plausible example to illustrate those abstract science-philosophical considerations (Arageorgis & Baltas, 1989; Auyang, 2003).

More than the factual theories, which objectively describe properties of classes of devices, the operational theories in the engineering disciplines are practice-oriented. Their purpose is the formulation of *rules* for the optimal course of action in the processes of technical implementations (Bunge, 1998b, chpt. 11; Vincenti, 1990). In spite of their technical sophistication, operational theories are entirely subject to the *social determinations* of their corresponding engineering activities. In other words: they reflect the social 'values' in relation to a 'technological ideal' conceived by the agent who had initially posed the problem and requested its solution (Arageorgis & Baltas, 1989, p. 223).

The factual and the operational theories of an engineering discipline, the appropriate methods and techniques as well as the practically relevant knowledge from the corresponding basic sciences, are combined into a structured collection which constitutes a conceptual and methodological framework which is specific to that engineering discipline (Arageorgis & Baltas, 1989, pp. 223-224). Consequently, the *'solution'* of a technical problem is not determined by purely intra-scientific factors the way the solution of a scientific problem is. Alternative choice options, which appear at every coarser or finer level of technical design, often require important *extra-scientific* decisions in which social 'values' and costs of many kinds are taken into account. The solution to a technical problem is thus ‘scientifically indeterminate’ inasmuch the factual theories of engineering, which support scientific attestation, cannot select by themselves the *desired* solutions. All they can do is to delimit the domain of those solutions which are scientifically possible (Arageorgis & Baltas, 1989, p. 226). In philosophical terms this open situation is subject to the ‘*hermeneutics* of engineering’ (Poser, 1998), which means that engineers are concerned with what is *unique and specific* more than with what is universal and general. At this point the classical distinction between ‘*nomothetic*’ and ‘*idiographic*’ disciplines (Windelband, 1900, 1998) becomes relevant again, though the engineering disciplines had not been given much attention in this classification scheme from the late 19th century. Now it turns out that the engineering disciplines are epistemologically characterized by a peculiar ‘mixture’ of both nomothetic and idiographic ‘elements’. In the realm of software construction, for example, all software systems are delimited scientifically by the undecidability of the halting problem, the NP-hardness of the knapsack problem, the acceleration limits described by Amdahl’s law, and the like; nonetheless there are still many possible and viable technical solutions to many practical problems within these strict computer-scientific limits.

All in all the foregoing paragraphs speak strongly against any possibility of any seriously acceptable ‘general theory’ (in a science-philosophically acceptable sense of the term ‘theory’) for any software-*related* engineering discipline which truly deserves the title ‘engineering’; (remember in this context that there exists no single ‘hardware engineering’ discipline for everything that is made of tangible materials). As a consequence of the considerations outlined above, such an (utopian) ‘general theory’ of software engineering would have to comprise a multitude of operational sub-theories which are in themselves extra-scientific and highly dependent on non-objective (hence: ever contestable) social factors. These factors distinguish all engineering disciplines from the pure sciences (Gruner, 2010; Gruner & Kroeze, 2014) in which ‘general’ theories can be formulated.

In other words: Because of the specific differences between science and engineering, the utopian search for a science-equivalent ‘general theory’ (like the ‘big bang’ theory of cosmology, or the Darwinian evolution theory of biology) in and for the domain of software engineering (which includes information systems design) does not make sense. Scientific theories are always strictly dis-interested, descriptive and predictive, whereas engineering as a whole is always motivated by varying vested interests (Bunge, 1998b, chpt. 11), and always comprises freely chosen, normative and voluntary features, which stand outside the scope of purely descriptive and predictive scientific theories. Such forms of knowledge have also been called ‘many-level theories’ which can only arise from ‘inter-disciplinary networks’ (Poser, 1998). Moreover, as previously mentioned: ‘software’ as a subject is far too broad and varied to make *one* genuine *engineering* discipline for it, (such as also no single one ‘hardware engineering’ discipline exists). As there are many different specialized ‘hardware engineering’ disciplines for different types of ‘hardware’ devices (trains, ships, cars, etc.), there must first arise different and diversified software engineering discipline*s* (in plural) for different types of software systems before the ‘craft’ of software construction can reach the envisaged status of ‘engineering’. Only then we can hope to define salient, non-trivial and fruitful engineering theories for these specialized sub-disciplines within the wide field of software construction. One positive example in this context is the already highly specialized sub-discipline of *compiler construction*, for which salient, computer-science-based, fruitful and highly applicable compiler construction 'micro theories' are already known since several decades.

5 THE MULTIPLICITY OF HIGHLY SPECIALIZED 'MICRO THEORIES' IN THE ENGINEERING DISCIPLINES

In contrast to other notions of 'micro theory' (Stol et al., 2016), which are rather fruitless notions, micro theories are *not* theories about one singular particular technical artifact in its 'idiographic' uniqueness. Rather, micro theories deal with *classes* of *similar* technical artifacts, which are also known as 'devices' (Vincenti, 1990), of which a singular particular technical artifact is only an instance. A well-known historic example of a 'micro theory' can be found in the development of the theory of heat-power-transmission systems (which is nowadays included in the physical sub-discipline of thermodynamics) during the era of the steam machines in the 'classical' period of industrialization.

Already in the 1970s, philosophers of science from the Institute of Starnberg had postulated that so-called *finalized sciences*, which receive their driving motivations extra-scientifically from societal problems rather than intra-scientifically from the sciences' own theoretical desiderata, tend to produce a flurry of highly specialized technological sub-theories on the basis of general scientific theories (Böhme et al., 1973). Though this Starnberg'ian science-historical meta-theory had been strongly criticized for its exaggeratedly 'uniform' treatment of all sciences as if they were one (Dahrendorf, 1976; Pfetsch, 1979; Rasmussen, 1982; Restivo, 1984), it is to a large extent acceptable as far as only the engineering sciences are concerned.

A 'seminal' introduction into the history, practice, and epistemology of engineering, which was written on the basis of case studies and examples from the discipline of *aeronautical* engineering (Vincenti, 1990), offered a noteworthy combination of factual experiences together with science-philosophical and methodological reflections at a higher level of abstraction. As far as *software* 'engineering' is concerned, especially the work of Jackson and Maibaum facilitated a fruitful transfer of some of Vincenti's most important insights into the field of software construction. Their interpretations of Vincenti's findings are by-and-large compatible with the analyzes provided by other philosophers of science and technology (Arageorgis & Baltas, 1989; Bunge, 1998b), as well as with analyzes provided by other engineering-oriented informaticians (Broy & Schmidt, 1999).

In the various areas of software construction, 'mainstream' *formal methods*,[5] which can be regarded as 'operational theories' (Arageorgis & Baltas, 1989), are useful only for small and narrowly defined classes of software development problems (Jackson, 1998). In such a context, universal methods cannot be effective because they cannot sufficiently capture the

[5]The term 'formal methods' as such is already quite old (Beth, 1962). Only much later this term was adopted in the same sense by the computing and software construction communities.

all-important particular details of the problems at hand. Especially, any 'universal' methods must 'abstract away' all those aspects of a software development task the uniform-universal treatment of which would be too difficult (Jackson, 1998, p. 192). Crucial in engineering is a proper *balance between what is general and what is particular*. An engineering *handbook*, for example, is *not* a compendium of abstract fundamental principles (Gruner et al., 2020): on the contrary, the engineering handbook contains a corpus of rules and procedures in the form of 'nomopragmatic statements' (Bunge, 1998b) through which those general abstract principles can be most efficiently applied to the particular design tasks at hand (Jackson, 1998, p. 193). In software development, too, the most useful application case for the precision which formality (i.e.: 'theoretical-ness' in the classical scientific sense of the notion of 'theory') can offer is the case of sharply focused 'micro methods' for very specific purposes. Thereby, which is quite compatible with the Starnberg'ian philosophy of 'finalization' (Böhme et al., 1973), each existing formal method may be expected to 'give birth' to several 'micro methods' in such a manner that only parts of the entire formal apparatus of their common 'parent method' are used. Those parts must be accompanied by explicit declarations about the specific contexts in which they are expected to be successfully applied (Jackson, 1998, p. 193). A specific recent *example* can be found in the *Formal Methods for the Railway Domain* (Gruner et al., 2013; Gruner et al., 2016; Gruner et al., 2020). The use of such 'micro methods' characterizes an *applied* (i.e.: not 'pure') scientific discipline (Bunge, 1998b, chpt. 11; Jackson, 1998, p. 193) in which large overarching claims are merely absurd (Jackson, 1998, p. 194).

Thus, domain-specificity is 'inherent' in successful engineering disciplines (Maibaum, 2009). As far as the above-mentioned transfer of insights from Vincenti is concerned, the '*theoretical tools*' category, which is one of several categories of engineering knowledge (Vincenti, 1990), is most important for the topic of this lecture. Those 'theoretical tools' are needed by engineers to 'underpin' their work. They include 'intellectual concepts' for thinking about design as well as mathematical methods and 'theorie*s*' (in plural) for making design-related calculations. As far as the above-mentioned difference between science and engineering is concerned, the engineer's 'theoretical tools' may be devised specifically for use in particular contexts of application and might thus possibly be of *no intellectual value for a pure scientist* or mathematician (Maibaum, 2009, p. 9). As a questionable *counter-example* in software engineering we may regard the notorious 'capability maturity model' (CMM) which does not directly address the development of the needed methods of engineering and is thus *too general* to be of immediate use to software engineers (Maibaum, 2009, p. 10). Accordingly, various 'in-the-small theories' are particularly needed in software construction for the purpose of representing a software system's run-time behavior (because only at run-time can software become effective and potentially dangerous), as well as for the purpose of representing important 'higher level' (or 'emergent') properties and qualities, which are difficult to quantify, such as safety, reliability, maintainability, and the like (Maibaum, 2000, p. 169). However, because those qualities (or, more precisely: our human perception thereof) will vary from application domain to application domain (for example: our ideas about what constitutes 'high reliability' might differ from each other when we compare aviation control software against software for playing games), the development of those 'in-the-small theories' can only be application-domain-specific. This observation will eventually lead to the systematic compilation of *many* handbooks (bodies of knowledge) for the typical software application domains such as banking and finance, tele-communication, industrial control, robotics, and the like (Maibaum, 2000, p. 169). They 'correspond' to the many device-specific branches of 'hardware engineering' which are known since the early days of the industrial era. Last but not least it must also be conceded that those application-domain-specific 'in-the-small theories' will most likely appear *initially* merely as *'guides' to aid the organization of practical knowledge* on the basis of a *weaker notion of 'theory'* which is not yet the same as the strict notion of 'theory' in its fully formalized mathematical-scientific sense (Maibaum, 2000, p. 171). Such an initial informality, however, might gradually 'shrink' with the growth of knowledge in the course of a discipline's historic development, during which *the anti-formalist attitude of the so-called 'practical men' will eventually disappear* (Baber, 1997).[6]

In other words: whilst the notion of 'theory' underlying the concept of domain-specific micro theories might initially sit on the 'vague' side of Seiffert's classification scheme, scientificness and formality will increase as an engineering discipline makes epistemic and epistemological progress during the course of its history (Baber, 1997). Never, however, are they 'general' in a 'physicalist' sense of 'generality' as envisaged by the leading members of the above-mentioned GTSE community.

6 THE TWO DIMENSIONS OF MICRO THEORY DEVELOPMENT IN SOFTWARE ENGINEERING

From the foregoing sections of this discussion the following intermediate conclusions may be drawn:

- According to Dijkstra's warning and relevant historical accounts of 'classical' engineering disciplines (Baber, 1997), the number of scientifically supported micro theories and their level of mathematical formalization must eventually increase (in spite of predictable resistance from some of the so-called 'practical men') in all branches and sub-branches of software 'engineering' (including: information systems) for our discipline to overcome its pre-engineering 'crafting' era as well as to get rid of the pseudo-scientific 'guru-ism' by which this era is typically characterized.

- At the same time, however, because of the specific differences between science and technology (engineering), the above-mentioned growth of scientifically supported theoretical formality cannot happen 'in general' (such as in the pure sciences), but must happen in ever more specific and particular sub-theories and sub-domains of application (Jackson,

[6]In reply to a question from the audience at the AISSAC'2020 an additional hint to Max Planck may suffice at this point, according to whom new insights will not get established by 'conversion' of their opponents but rather by the old-age-retirement of their opponents when a new generation of young researchers takes its place.

1998; Maibaum, 2000, 2009). For comparison: a general discipline of 'hardware engineering' (including everything from nano-mechanics to giant ship yards) does not and cannot exist either.

- Moreover: the *technological* theories, by which the engineering disciplines are characterized, are to a large extent *operational* theories which the facts-oriented and 'disinterested' pure sciences do not possess to such a large extent (Arageorgis & Baltas, 1989; Bunge, 1998b; Vincenti, 1990).

As a consequence of these three points the multiplicity of software engineering's formalized (or at least: to-be-formalized) micro theories can be systematically ordered in the following two dimensions.

Operational: These micro theories are related to those *methods and activities* of software engineering which re-occur in all (or most) software engineering projects and are thus independent of their sub-domain of application. In this category we can find the (formalized) micro theories of mereological domain analysis, requirements engineering, program verification, software testing, complexity analysis, software refactoring, and the like. Systematic teach-and-study books for academics and students in each of those areas of activity are already widely available.

Substantial: These micro theories are (or will be in the future) related to the *domains and subdomains of application* for and in which software is typically developed and deployed. In this category we can (or will be able to) find the (formalized) micro theories of classical areas such as compiler construction, database management systems, operating systems, as well as yet-to-be-developed domain theories for other 'typical' classes of software such as: medical information systems (Silva et al., 2018), finance and accounting software, natural language processors, robot software, and many more. They correspond (by analogy) to the above-mentioned device-class-specific branches of 'hardware engineering' (for which no 'general theory of hardware engineering' can possibly exist).

Because of the two-dimensionality of this methodological and meta-theoretical systematization, micro-theoretical improvements and refinements can be obtained via pair-wise combinations, such that it will eventually be possible to rely (for example) on specific micro theories for 'testing of compilers', 'verification of operating systems', 'modular specification of robotic software', re-factoring of DBMS, requirements elicitation for accounting software, and the like, which are all specific instances of the 'divide-and-conquer' strategy (meta-method) for the 'simplification of structural complexities'. As far as the intended mathematical-logical formalization of all those micro theories is concerned, good progress in the growth of knowledge has already been made (for example) in the following areas:

- So-called *ontologies* (Kroeze, 2010), by means of which the 'domains of discourse' for many typical software applications can be defined, are amenable to formalization by *description logics* with known decidability results and logically correct inference rules (Baader et al., 2007). Closely related to them are the logical techniques of *mereology* by means of which part-whole-relations of systems can be formally described (Bjørner & Eir, 2010). These micro theories can also be applied to 'information systems' or 'knowledge systems' such as expert systems or deductive data bases.

- *Deontic logic*, by means of which we can express formally what is 'mandatory', 'allowed', or 'forbidden', has been made fruitful in requirements engineering especially for the stipulation of service contracts (Prisacariu & Schneider, 2012).
- *Complexity theory*, one of the best-researched topics in theoretical informatics, enables scientifically grounded performance predictions of almost any 'typical' algorithm in its context of application which can lead directly to the formulation of empirically falsifiable hypotheses for the purpose of experimental performance evaluations (Guéneau, 2019).[7]
- Empirically falsifiable hypotheses about the to-be-expected benefits of parallelization can be obtained from considerations similar to Amdahl's law (Amdahl, 1967), which is part of a 'micro theory' in the computer engineering domain. Indeed Amdahl's parallelization law is of 'phenomenological-descriptive' type, which is quite typical for the laws of engineering, similar to Kirchoff's electrical circuit laws in electrical engineering which are also 'phenomenological-descriptive'.
- The mature status of theory-guided *software testing* as a 'scientifically attested technology' (Arageorgis & Baltas, 1989) was recently confirmed science-philosophically (Angius, 2013, 2014) with reference to relevant micro theories (Ammann & Offutt, 2008).
- The application of *formal language theory* and *automata theory* in the field of compiler construction is already so well understood that many parts of a compiler software can nowadays be produced fully automatically by means of commercially available meta-compilers (lexer generators, parser generators, and the like). Thanks to those micro theories, compilers (as special instances of software systems) have thus already reached the status of what was elsewhere called 'devices' (Vincenti, 1990).
- *Relational algebra* is the micro theory behind the construction and operation of many commercially available database management systems.

Further examples of formalized micro theories ought to be developed and provided for many other more-or-less well-established sub-areas of software construction.

For the status and reputation of the software 'engineering' discipline it would indeed be good if a larger proportion of all software constructing practitioners would actually and seriously use those already available and applicable formalized or semi-formalized micro theories

[7]There are some 'wicked' algorithmic systems which exhibit 'nasty' worst-case performance for only very few 'unlikely' (though still legal) inputs: malicious users with knowledge thereof, however, could de-facto 'knock out' such systems by deliberately choosing those 'unlikely' inputs.

instead of continuing their profession as mere 'crafting' under the influence of 'gurus' while waiting for the notorious 'software crisis' to be solved by a mythical-utopian 'general theory' in an indefinite future.

Last but not least: when we take the typical knowledge transfer time of approximately twenty years from basic (foundational) research to standardized industrial practice into account, the entire software production industry of nowadays 'should' already have reached in many of its sub-fields the 'micro theoretical' maturity of the year 2000, whereby some pre-theoretical 'crafty' residuals must be tolerated as previously described and discussed (Arageorgis & Baltas, 1989).

7 SUMMARY AND OUTLOOK

Not every collection of knowledge is a proper theory. 'General' theories 'of' software engineering (as a whole) cannot exist, such that any quest for them (such as pursued by the GTSE community) is a futile waste of intellectual resources. Operational as well as substantial micro theories for the various branches and sub-branches of 'software engineering' (including information systems) are still lacking and ought to be developed as a matter of urgency, for the solidification of our still too 'crafty' discipline. With the typical turn-around-time of 20 years from research to practice, micro-theoretical innovations or improvements which we make today might become 'common practice' not before the year 2040.

Acknowledgments and Dedications: Many thanks to *Dines Bjørner*, *Michael Jackson*, as well as *Tom Maibaum* for several inspiring conversations related specifically to the topics of this contribution. Thanks also to *Manfred Nagl* for a sequence of earlier conversations about the essence of engineering and the deep hermeneutic connections between 'understanding' and 'implementing' in the discipline of software construction. Thanks to the editors of the *Zeitschrift für allgemeine Wissenschaftstheorie* for their helpful provision of a highly relevant literature reference (Arageorgis & Baltas, 1989). Another interesting literature reference (Oates, 2006) was kindly sent to me by *Jan Kroeze*. Much inspiration I have also received from ongoing conversations with *Martin Olivier*, whose informatics- and software-related discipline of digital forensics is also still struggling with meta-theoretical questions concerning the discipline's theoretical-ness (or shortage thereof). He has, moreover, provided much-appreciated technical help with a rapid internet connection to the 'online meeting' of AISSAC'2020: without his technical aid I would not have been able to participate in this event. Thanks to the members of the ACM's SEWORLD network for having provided the triggering motivation for my invited lecture. Thanks also to the participants of the IACAP'2019 and AISSAC'2020 meetings for their interesting questions and remarks which enabled me to develop my thoughts from earlier considerations (Gruner, 2019) into the form of this full-length 'invited lecture' contribution. Many thanks to *Maureen Tanner* for having invited me to speak at AISSAC'2020, as well as to the editors of SACJ (this journal) for their willingness to publish this full-length text of my

invited lecture in the form of a non-reviewed ‘opinion contribution’.

This contribution is dedicated to the memory of *John Conway* and *Ken MacGregor*, two noteworthy mathematicians and informaticians, who have alas passed away in this year’s (2020) global COVID19 pandemic. May they be remembered through many citations.

References

Amdahl, G. (1967). Validity of the single processor approach to achieving large scale computing capabilities. *AFIPS Spring Joint Computer Conference*. https://doi.org/10.1145/1465482.1465560

Ammann, P. & Offutt, J. (2008). *Introduction to software testing*. Cambridge University Press.

Angius, N. (2013). Model-based abductive reasoning in automated software testing. *Logic Journal of the IGPL*, *21*(6), 931–942. https://doi.org/10.1093/jigpal/jzt006

Angius, N. (2014). The problem of justification of empirical hypotheses in software testing. *Philosophy and Technology*, *27*(3), 423–439. https://doi.org/10.1007/s13347-014-0159-6

Arageorgis, A. & Baltas, A. (1989). Demarcating technology from science — Problems and problem solving in technology. *Zeitschrift für allgemeine Wissenschaftstheorie*, *20*(2), 212–229.

Aravena-Reyes, J. (2018). Métis — Reconfiguring the philosophy of engineering. In A. Fritzsche & S. Oks (Eds.), *The future of engineering — Philosophical foundations, ethical problems and application cases* (pp. 123–136). Springer.

Auyang, S. (2003, January 21). Why did chemical engineering emerge in America instead of Germany? Lecture notes of a lecture presented at the Eidgenössische Technische Hochschule Zürich. http://www.creatingtechnology.org/eng/chemE.pdf

Baader, F., Calvanese, D., McGuinness, D., Nardi, D. & Patel-Schneider, P. (Eds.). (2007). *The description logic handbook* (Second). Cambridge University Press.

Baber, R. (1997). Comparison of electrical ‘engineering’ of Heaviside’s times and software ‘engineering’ of our times. *IEEE Annals of the History of Computing*, *19*(4), 5–16. https://doi.org/10.1109/85.627895

Bamm, P. (1956). *Ex ovo — Essays über die Medizin*. Deutsche Verlags-Anstalt.

Beth, E. (1962). *Formal methods*. Springer.

Bjørner, D. & Eir, A. (2010). Compositionality: Ontology and mereology of domains — Some clarifying observations in the context of software engineering. *LNCS*, *5930*, 22–59.

Bochenski, I. (1954). *Die zeitgenössischen Denkmethoden*. Francke.

Bochenski, I. (1965). *The methods of contemporary thought*. Reidel.

Böhme, G., van den Daele, W. & Krohn, W. (1973). Die Finalisierung der Wissenschaft. *Zeitschrift für Soziologie*, *2*(2), 128–144. https://doi.org/10.1515/zfsoz-1973-0202

Bristol, T. (2018). The engineering knowledge research program. In A. Fritzsche & S. Oks (Eds.), *The future of engineering — Philosophical foundations, ethical problems and application cases* (pp. 69–88). Springer. https://doi.org/10.1007/978-3-319-91029-1_5

Brooks, F. (1995). *The mythical man-month — Essays on software engineering* (Anniversary). Addison Wesley.

Broy, M. & Schmidt, J. (1999). Informatik: Grundlagenwissenschaft oder Ingenieurdisziplin? *Informatik Spektrum*, *22*(3), 206–209. https://doi.org/10.1007/s002870050139

Bunge, M. (1998a). *Philosophy of science* (Revised, Vol. 1 — From Problem to Theory). Transaction.

Bunge, M. (1998b). *Philosophy of science* (Revised, Vol. 2 — From Explanation to Justification). Transaction.

Charpa, U. (1996). *Grundprobleme der Wissenschaftsphilosophie*. Schöningh.

Coy, W. (1992). Für eine Theorie der Informatik. In W. Coy, F. Nake, J. Pflüger, D. Siefkes & R. Stransfeld (Eds.), *Sichtweisen der Informatik — Theorie der Informatik* (pp. 17–32). Vieweg. https://doi.org/10.1007/978-3-322-84926-7_3

Coy, W. (1993). Reduziertes Denken — Informatik in der Tradition des formalistischen Forschungsprogramms. In P. Schefe, H. Hastedt, Y. Dittrich & G. Keil (Eds.), *Informatik und Philosophie* (pp. 31–52). B·I· Wissenschaftsverlag.

Dahrendorf, R. (1976, May 21). Die Unabhängigkeit der Wissenschaft — Vorläufiges Schlußwort in einer wichtigen Debatte. *Die Zeit 22*.

Exman, I., Perry, D., Barn, B. & Ralph, P. (2016). Separability principles for a general theory of software engineering — Report on the GTSE 2015 workshop. *ACM SIGSOFT Software Engineering Notes*, *41*(1), 25–27. https://doi.org/10.1145/2853073.2853093

Freudenthal, H. (Ed.). (1961). *The concept and the role of the model in mathematics and natural and social sciences — Proceedings of the colloquium sponsored by the Division of Philosophy of Sciences of the International Union of History and Philosophy of Sciences organized at Utrecht, January 1960*. Springer. https://doi.org/10.5840/philstudies19631264

Galloway, A. (2004). *Protocol — How control exists after decentralization*. MIT Press.

Götschl, J. (1980). Theorie. In J. Speck (Ed.), *Handbuch wissenschaftstheoretischer Begriffe* (pp. 636–646). Vandenhoeck & Ruprecht.

Greeley, A. (1975). Der Verrat des Intellektuellen. *Concilium*, *11*(1), 30–36.

Gregor, S. (2006). The nature of theory in information systems. *MIS Quarterly*, *30*(3), 611–642. https://doi.org/10.2307/25148742

Gruner, S. (2010). Software engineering between technics and science — Recent discussions about the foundations and the scientificness of a rising discipline. *Zeitschrift für allgemeine Wissenschaftstheorie*, *41*(1), 237–260. https://doi.org/10.1007/s10838-010-9116-y

Gruner, S. (2016). Heinz Zemanek's almost forgotten contributions to the early philosophy of informatics. Paper #1. *ACIS'16 Proceedings of the 27th Australasian Conference on Information Systems*.

Gruner, S. (2019). Inappropriate notions of 'theory' and their practical consequences in the discipline of software 'engineering' — Discussion abstract. University of Pretoria technical report. https://repository.up.ac.za/handle/2263/70340

Gruner, S., Haxthausen, A., Maibaum, T. & Roggenbach, M. (2013). Towards a formal methods body of knowledge for railway control and safety systems — FM-RAIL-BOK workshop 2013. DTU Denmark technical report 20.

Gruner, S. & Kroeze, J. (2014). On the shortage of engineering in recent information systems research. *ACIS'14 Proceedings of the 25th Australasian Conference on Information Systems*.

Gruner, S., Kumar, A. & Maibaum, T. (2016). Towards a body of knowledge in formal methods for the railway domain — Identification of settled knowledge. *CCIS*, *596*, 87–102. https://doi.org/10.1007/978-3-319-29510-7_5

Gruner, S., Kumar, A., Maibaum, T. & Roggenbach, M. (2020). *On the construction of engineering handbooks — With an illustration from the railway safety domain*. Springer. https://doi.org/10.1007/978-3-030-44648-2

Guéneau, A. (2019). *Mechanized verification of the correctness and asymptotic complexity of programs — Doctorat d'Informatique*. "Diderot" Université de Paris 7.

Hall, J. & Rapanotti, L. (2017). A design theory for software engineering. *Information and Software Technology*, *87*(1), 46–61. https://doi.org/10.1016/j.infsof.2017.01.010

Heidelberger, M. (1993). Was erklärt uns die Informatik? Versuch einer wissenschaftstheoretischen Standortbestimmung. In P. Schefe, H. Hastedt, Y. Dittrich & G. Keil (Eds.), *Informatik und Philosophie* (pp. 13–30). B·I· Wissenschaftsverlag.

Hellige, H. (2008). Wissenschaft vs. Design — Konstruktionslehren für den Maschinenbau, den Computer und die Software im historischen Diskursvergleich. In M. Warnke & D. Weber-Wulff (Eds.), *Kontrolle durch Transparenz — Transparenz durch Kontrolle — Tagung des Fachbereich Informatik und Gesellschaft der Gesellschaft für Informatik* (pp. 113–125).

Hellige, H. (2012). Die Dialektik der informationellen Aufklärung — Ein Rückblick auf den Theoriediskurs von 'Informatik & Gesellschaft'. In C. Kühne, R. Rehak, A. Knaut, S. Ullrich, C. Kurz & J. Pohle (Eds.), *Per Anhalter durch die Turing-Galaxis* (pp. 55–60). Monsenstein und Vannerdat.

Hoffmeister, J. (Ed.). (1955). *Wörterbuch der philosophischen Begriffe* (Second). Felix Meiner.

Hübner, K. (1978). *Kritik der wissenschaftlichen Vernunft*. Karl Alber.

Hübner, K. (1983). *Critique of scientific reason*. University of Chicago Press.

Jackson, M. (1998). Formal methods and traditional engineering. *Journal of Systems and Software*, *40*, 191–194. https://doi.org/10.1016/S0164-1212(97)00165-9

Johnson, P. & Ekstedt, M. (2016). The tarpit — A general theory of software engineering. *Information and Software Technology*, *70*, 181–203. https://doi.org/10.1016/j.infsof.2015.06.001

Johnson, P., Ekstedt, M., Goedicke, M. & Jacobson, I. (2015). Towards general theories of software engineering [editorial]. *Science of Computer Programming*, *101*, 1–5. https://doi.org/10.1016/j.scico.2014.11.005

Kline, R. (2006). Cybernetics, management science, and technology policy — The emergence of 'information technology' as a keyword 1948-1985. *Technology and Culture*, *47*(3), 513–535.

Kroeze, J. (2010). Ontology goes postmodern in ICT. *Proceedings SAICSIT'10*, 153–159. https://doi.org/10.1145/1899503.1899520

Maibaum, T. (2000). Mathematical foundations of software engineering — A roadmap. *Proceedings Future of Software Engineering*, 161–172. https://doi.org/10.1145/336512.336548

Maibaum, T. (2009). Formal methods versus engineering. *Inroads SIGCSE Bulletin*, *41*(2), 6–11. https://doi.org/10.1145/1595453.1595455

Margalit, A. (1996). *The decent society*. Harvard University Press.

Naur, P. (1985). Programming as theory-building. *Microprocessing and Microprogramming*, *15*(5), 253–261. https://doi.org/10.1016/0165-6074(85)90032-8

Nowotny, H., Scott, P. & Gibbons, M. (2001). *Re-thinking science — Knowledge and the public in an age of uncertainty*. Polity.

Oates, B. (2006). *Researching information systems and computing*. Sage.

Pfetsch, F. (1979). The 'finalization' debate in Germany — Some comments and explanations. *Social Studies of Science*, *9*(1), 115–124. https://doi.org/10.1177%2F030631277900900107

Pirtle, Z., Odenbaugh, J. & Szajnfarber, Z. (2018). 'The one, the few or the many?' — Using independence as a strategy in engineering development and modeling. In A. Fritzsche & S. Oks (Eds.), *The future of engineering — Philosophical foundations, ethical problems and application cases* (pp. 13–31). Springer. https://doi.org/10.1007/978-3-319-91029-1_2

Polanyi, M. (2009). *The tacit dimension* (Revised). University of Chicago Press.

Poser, H. (1998). On structural differences between science and engineering. *VirginiaTech Electronic Journals — Society for Philosophy and Technology*, *4*(2). https://doi.org/10.5840/techne1998426

Prisacariu, C. & Schneider, G. (2012). A dynamic deontic logic for complex contracts. *Journal of Logic and Algebraic Programming*, *81*(4), 458–490. https://doi.org/10.1016%2Fj.jlap.2012.03.003

Rasmussen, S. (1982). Finalization and completed theories. *Zeitschrift für allgemeine Wissenschaftstheorie*, *13*(2), 359–369.

Restivo, S. (1984). Finalization — Cool radicalism versus the republic of science. *4S Review*, *2*(4), 14–20.

Schiaffonati, V. (2018). Philosophy of engineering and the quest for a novel notion of experimentation. In A. Fritzsche & S. Oks (Eds.), *The future of engineering — Philosophical foundations, ethical problems and application cases* (pp. 89–103). Springer. https://doi.org/10.1007/978-3-319-91029-1_6

Schlette, H. (1975). Die Autorität des Intellektuellen. *Concilium*, *11*(1), 24–30.

Schöning, U. (1992). *Logik für Informatiker* (Third). B·I· Wissenschaftsverlag.

Seiffert, H. (1992). Theorie. In H. Seiffert & G. Radnitzky (Eds.), *Handlexikon zur Wissenschaftstheorie* (pp. 368–369). DTV.

Silva, E., Bartholo, R. & Proenca, D. (2018). Managing the state of the art of engineering — Learning from medicine. In A. Fritzsche & S. Oks (Eds.), *The future of engineering — Philosophical foundations, ethical problems and application cases* (pp. 217–227). Springer. https://doi.org/10.1007/978-3-319-91029-1_15

Stol, K., Goedicke, M. & Jacobson, I. (2016). Introduction to the special section — General theories of software engineering — New advances and implications for research. *Information and Software Technology*, *70*, 176–180. https://doi.org/10.1016/j.infsof.2015.07.010

Vincenti, W. (1990). *What engineers know and how they know it — Analytical studies from aeronautical history*. John Hopkins University Press.

Wang, N. & Li, B. (2018). Three stages of technical artifacts' life cycle — Based on a four factors theory. In A. Fritzsche & S. Oks (Eds.), *The future of engineering — Philosophical foundations, ethical problems and application cases* (pp. 113–122). Springer. https://doi.org/10.1007/978-3-319-91029-1_8

Wilber, K. (2000). *A theory of everything*. Shambhala.

Windelband, W. (1900). *Geschichte und Naturwissenschaft — Rede zum Antritt des Rectorats der Kaiser-Wilhelms-Universität Strassburg, gehalten am 1. Mai 1894* (Second). J.H.E. Heitz & Mündel.

Windelband, W. (1998). History and natural science. *Theory and Psychology*, *8*(1), 5–22. https://doi.org/10.1177%2F0959354398081001

Winsberg, E. (2010). *Science in the age of computer simulation*. University of Chicago Press.

Young, M. (2018). Intuition and ineffability — Tacit knowledge and engineering design. In A. Fritzsche & S. Oks (Eds.), *The future of engineering — Philosophical foundations, ethical problems and application cases* (pp. 53–67). Springer. https://doi.org/10.1007/978-3-319-91029-1_4