Inappropriate Notions of 'Theory' and their Practical Consequences in the Discipline of Software 'Engineering': DISCUSSION ABSTRACT

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

In my discussion contribution to **IACAP`2019** (Mexico City, June the 6th, 2019) I argue that the quest for a so-called 'general theory of software engineering' is, all-in-all, not feasible. In addition to supporting this negative assertion with meta-theoretical arguments from philosophy of science, I also argue positively that software 'engineering' may well become 'proper' engineering via the development and appropriate application of a multitude of domain-specific 'micro theories', by which successful engineering disciplines are typically characterised. Since different meta-theoretical opinions exist even about the question of what is a 'micro theory', I will also have to explicate which definition of the notion of 'micro theory' will be most appropriate in and for this particular domain of discourse.

In July 2014, the following message was 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"*.

Topically related to the above-mentioned communication are a number of well-organised scholarly activities [**1**], including (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 emphasised, 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?"* [**2**]. In addition to such search for 'generality' (i.e.: a classical, science-oriented approach to the matter) it is also typical for these kind of efforts that they rarely problematise 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, and no definition of the term 'theory' is given. The 'GTSE' workshop of 2015 ended with notable sentiments of frustration [**2**] —which are at least in part also due to some intra-community faction differences between the 'engineers' and the 'sociologists' with their different epistemological and meta-theoretical points of view [**3**]— however nobody seems to have asked whether such frustration was perhaps a necessary consequence of the impossibility to solve an inherently insoluable problem (like the proveriabl attempt at 'squaring the circle'). Although the participants of 'GTSE 2015' already had the correct insight —*"based upon philosophy of science and software engineering practice"*— *"that engineering fundamentally differs from scientific disciplines"* [**2**], nobody seems to have consequently questioned the 'feasibility' of a science-like 'general theory' for a 'fundamentally different' engineering discipline.

In [4] a so-called 'design theory' for (not: 'of') software engineering is presented which is meant to be *"a theory that characterises the elements of a software problem solving in terms of the effect they have on the process of design"*. Also in the case of [4] the basic notion of 'theory' itself is not precisely defined and more-or-less being taken for granted, although a distinction between 'product' theories (about the things made) and 'production' theories (about the work-steps that lead to the things made) is taken into account. Very importantly it is acknowledged in [4] that *"the community of also trying to come to terms with what is meant by >>theory<<"*. Everywhere in [4] we can find references to a notion of 'theory' by [5] which, however, stems from the field of 'information systems' (IS) and is thus (like the entire field of IS) notably sociology-influenced and business-management-oriented: for particular historic reasons [6] the IS community does typically not maintain a self-view of (or as) a community of engineering [3][7].

In the year 2016, a 'Special Section on General Theories of Software Engineering' [8] was published by the Journal for Information and Software Technology. The most interesting paper in that special section is [3], as it refers with emphasis to a *general* theory (whilst the other papers of that special section merely refer to specialized theories of smaller aspects and sub-aspects within software engineering). In their guest-editorial preface, which also contains several further noteworthy literature references, the guest-editors of that special issue referred once again to *physics* as the reference discipline for general theories [8] — thus tacitly implying that that an engineering discipline (like software engineering) would (or should) ultimately have to be(come) something like a quasi-physical or quasi 'nature-scientific' discipline. Such a (tacit) physicalist attitude ignores the simplity 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 climatology, physics is equally at loss as far as the production of substantial, non-trivial 'general' theories is concerned. Moreover, the notion of 'theory' expressed in [8] cannot be seriously defended science-philosophically at that point where a 'micro theory' is identified with merely a small number of hypotheses concerning some technical properties of one particularly given software system (software product) [8]. Such a *device*-specific notion of 'micro theory', however, is not consistent with the *area*-specific notion of 'micro theory' in the established engineering disciplines. All in all the notion of 'theory' in [8] remained as vague as it had always been in the 'GTSE' community.

From a further elaboration of these problems and issues (which must be omitted here due to lack of space) the following conclusions can be drawn: According to Dijkstra's warning and Baber's historical account of 'classical' engineering disciplines [9], the level of mathematical formalisation must eventually increase in all branches and sub-branches of software 'engineering', in order to overcome the discipline's pre-scientific 'crafting' era, as well as to get rid of the pseudo- scientific 'guru-ism' by which the pre-scientific 'crafting' practices are often accompanied. At the same time, however, because of the specific differences between science and engineering (technology), the above-mentioned growth of theoretical formality cannot happen 'in general' (such as in the pure sciences), but must happen in the context of ever more specific and particular sub-theories and sub-domains of application [10][11][12]. For comparison: a general discipline of 'hardware engineering', including everything from nano-mechanics to giant ship yards, does not (and cannot) exist. Moreover: according to Arageorgis and Baltas [13], as well as Bunge [14] and Vincenti [15], the technological theories, by which the engineering disciplines are characterised, are to a large extent *operational* theories which the fact-oriented sciences do not possess in such a strong form.

A full-paper on the basis of this IACAP`2019 Discussion Abstract is in preparation. Thanks to the participants of the conference for their insightful comments and remarks after my talk in Mexico-City on the 6th of June 2019.

References

[1] P. Johnson, M. Ekstedt, M. Goedicke, I. Jacobson: *Editorial— Towards General Theories of Software Engineering*. Science of Computer Programming 101, pp. 1-5, 2015.

[**2**] I. Exman, D. Perry, B. Barn, P. Ralph: *Separability Principles for a General Theory of Software Engineering: Report on the GTSE 2015 Workshop*. ACM SigSoft Software Engineering Notes 41/1, pp. 25-27, 2016.

[**3**] P. Johnson, M. Ekstedt: *The Tarpit: A General Theory of Software Engineering*. Information and Software Technology 70, pp. 181-203, 2016.

[**4**] J. Hall, L. Rapanotti: *A Design Theory for Software Engineering*. Information and Software Technology 87/1, pp. 46-61, 2017.

[**5**] S. Gregor: *The Nature of Theory in Information Systems*. MIS Quarterly 30/3, pp. 611-642, 2006.

[**6**] R. Kline: *Cybernetics, Management Science, and Technology Policy: the Emergence of 'Information Technology' as a Keyword 1948-1985*. Technology and Culture 47/3, pp. 513-535, 2006.

[**7**] S. Gruner, J. Kroeze: *On the Shortage of Engineering in Recent Information Systems Research*. ACIS'14 Proceedings of the 25th Australasian Conference on Information Systems, Auckland, 2014.

[**8**] K. Stol, M. Goedicke, I. Jacobson: *Introduction to the Special Section: General Theories of Software Engineering: New Advances and Implications for Research*. Information and Software Technology 70, pp. 176-180, 2016.

[**9**] R. Baber: Comparison of Electrical 'Engineering' of Heaviside's Times and Software 'Engineering' of our Times. IEEE Annals of the History of Computing 19/4, pp. 5-16, 1997.

[**10**] M. Jackson: *Formal Methods and Traditional Engineering*. Journ. Syst. Software 40, pp. 191-194, 1998.

[**11**] T. Maibaum: *Mathematical Foundations of Software Engineering: a Roadmap*. Proceedings Future of Software Engineering, pp. 161-172, ACM Press, 2000.

[**12**] T. Maibaum: *Formal Methods versus Engineering*. Inroads SIGCSE Bulletin 41/2, pp. 6-11, 2009.

[**13**] A. Arageorgis, A. Baltas: *Demarcating Technology from Science: Problems and Problem Solving in Technology.* Zeitschrift f. allgem. Wissenschaftstheorie XX/2, pp. 212-229, 1989.

[**14**] M. Bunge: *Philosophy of Science 2: From Explanation to Justification*. Transaction Publ., rev. ed., 1998.

[**15**] W. Vincenti: *What Engineers know and How they know it: Analytical Studies from Aeronautical History*. John Hopkins Univ. Press, 1990.