Letter to the Editor: 40 Years after Garmisch

Stefan Gruner

Department of Computer Science, University of Pretoria, Republic of South Africa

40 Years ago: 1968! While Woodstock-inspired hippies, with long hair, made love, not war, and while leftwing students, successfully rebelled for a more liberal society in general and against the academic establishment in particular,[1] an event took place almost unnoticed, in a small Bavarian town at the foot of the Alps, which —in retrospective— turned out to be as important for our profession as the students' rebellion and the "Spirit of 68" was for the general liberalisation and opening up of the Western societies. That event was the now famous 1968 NATO Science Conference in Garmisch, Bavaria, at which the term SOFTWARE ENGINEERING was coined,[2] and the birth of a new profession was declared. Throughout the world of computer science and software engineering that event is being commemorated this year.[3]

In this letter, I do not want to repeat the history of software engineering which other, more competent people have so aptly described and discussed.[4] In spite of having fallen victim to the recent —now already notorious— "Heathrow Hiccup" at the new T5 terminal to London's Heathrow airport, in which software defects might have played their role (though human incompetence was probably to be blamed even more than technical defects), I also want to refrain from repeating the usual litany of failures that constitute the notorious "Software Crisis" — a term that was also coined a generation ago, when the financial cost of software started to exceed the financial cost of hardware —what a shock— and this cost relation between software and hardware has never been reversed since then.

Instead, I want to muse a little about the second word of the term, "engineering", because, whilst some people tend to believe that we in software engineering are still, basically, "craftsmen" who have not yet even approached "engineering" status, organisations like the EASST already speak about software "science" and software "technology"[5] — two little words about which volumes and volumes of philosophical writings have already been published.

The reason why I want to muse a little bit about software "craft" versus software "engineering" versus software "science" is to highlight the relation of these three concepts to what we teach at university and how we teach it, how much emphasis we place on each part of this triplet, etc. I recently discussed such questions with a colleague in Britain — which was, by the way, also one of the reasons why I got stuck in that already mentioned "Heathrow Hiccup" at London's T5 at all. Trying to be a good software engineer (whatever that means in detail), I try to explain the issue at hand in a "layered architecture" approach — though I will refrain from declaring which one of these three layers would be the "top" layer and which one would be the "bottom" layer; these are sensitive academic issues, and one does not wish to step on anybody's toes (as far as this is possible in a densely crowded room). Anyway, we now have the "scientific layer", the "engineering layer" and the "crafts layer" to think about — not to mention the "arts" (of programming, etc.), which are often quoted in this context, too.

Let's take an algorithm A. At the purely scientific layer we might not even know (and certainly would not care) whether A is useful for any purpose, but we would know that A's complexity is $\mathcal{O}(\mathcal{X})$ in the best case, $\mathcal{O}(\mathcal{Y})$ in the average case, and $\mathcal{O}(\mathcal{Z})$ in the worst case. Moreover we would not only "know" that this is all the case; we would also be able to *prove* it mathematically, from first principles, and thus achieve "knowledge" in the old Aristotelian sense of the word.[6]

At the engineering layer we would know that two variants, A and A', of our algorithm could be used for solving a given *external* problem P.[7] We would also know the best case, average case, and worst case complexities $\mathcal{O}(\mathcal{X})$, $\mathcal{O}(\mathcal{X}')$, $\mathcal{O}(\mathcal{Y})$, $\mathcal{O}(\mathcal{Y}')$, $\mathcal{O}(\mathcal{Z})$ and $\mathcal{O}(\mathcal{Z}')$ of these two algorithms, according to which we could make a well-informed decision about which algorithm to choose, A or A', as a "good" or "acceptable" or "elegant" solution to our given problem P. However, we would perhaps not know how to prove all those $\mathcal{O}$-properties mathematically from first principles; instead we would look up the results from a table in our standard engineering handbook. By the

Email: Stefan Gruner `stefan@cs.up.ac.za`

[1] Their battle cry in Germany was: *"Unter den Talaren, der Muff von tausend Jahren!"* — under the gowns, the reek of thousand years!

[2] P. Naur and B. Randell (Eds.): *Software Engineering — Report on a Conference sponsored by the NATO Science Committee.* Garmisch, Germany, 1968, published in January 1969.

[3] For example at ICSE, see *http://icse08.upb.de/program/40years.html*

[4] A. Brennecke and R. Keil-Slawik (Eds.): *Position Papers for Dagstuhl-Seminar 9635 on the History of Software Engineering.* Schloss Dagstuhl, Germany, 1996.

[5] European Association for Software Science and Technology, see *http://www.easst.org/*

[6] Generality and neccessity

[7] A so-called "real world" problem, in contrast to a purely intra-scientific problem

way, the word "engineer" itself is interesting enough; its French translation is "ingenieur", and, strangely enough, these two words, "engineer" and "ingenieur", sound quite similar, phonetically, from a listener's standpoint — yet their meanings and connotations point into different, almost opposite directions: "engineer" points to the engine, the machine, whereas "ingenieur" points to the genius, the mind behind the machine.

At the crafts layer,[8] finally, we would probably not even know anything about complexity and $\mathcal{O}$-properties at all, but we would know about the existence and the purpose of algorithm A, and we could also beautifully "code" it in a variety of techniques, iteratively or recursively, in a variety of programming languages, Java or Lisp or whatever, for a variety of platforms, Linux, Windows, and so on.

Now, where are we, as academic software "engineers", located in this 3-layered "architecture of knowledge", 40 years after the NATO Science Conference at Garmisch? Where is the "link" between software engineering, computer science, and related disciplines? Is (or should be) software engineering still a sub-area of computer science, or is (or should it be) already a fully-fledged discipline in its own right?[9] Moreover: Where in this 3-layered "architecture of knowledge" are (or should be) the universities, where are (or should be) the technical or vocational colleges and job academies? The answers to these questions do not only depend on political decisions being made in those kind of committees whose members usually enjoy the privilege of a permanently reserved sheltered car parking space, but also on the individual students coming to us — which brings me (eventually!) to my final theme for this letter.

Recently, I have been re-reading Confucius' *Conversations*, in translation of course, not in the two-thousand-and-something year old Chinese original. There we can find the following verdict about a student Dsai Yu.[10] What had he done? The verse reads as follows: "Dsai Yu once slept during the day. The master spoke: 'You cannot carve anything from rotten wood, and you do not paint onto a wall made of mud. It is not worth the effort to criticize Dsai Yu'!" More recently, a student came to visit me in my office, declaring that he had not understood something from my lecture, demanding further explanations and help. I asked the student: "Which books from the library have you already consulted, in order to find an explanation to your problem?" It turned out that this student had not even bothered to walk those hundred steps across our magnificent campus to our excellent and well-equipped academic library. Now, referring back to our software engineering conversation of above: on what "layer" of our 3-strated "architecture of knowledge" would you try to approach this guy? On the "scientific" layer? On the "engineering" layer? On the "crafts" layer? And would Confucius even call such a student a "student"? Let us hear Confucius himself again:[11] "The master spoke: 'I do not teach him, who is not committed and not busy in his attempts. I do not open the mind of him who does not open his mouth. And after I pointed to the one corner: he, who is then not able to point to the three other corners — I will not teach him any more'. "

I wonder if *such* a policy would have spared the world the experience of the "Heathrow Hiccup"?

[8] A. Arageorgis and A. Baltas: *Demarcating Technology from Science — Problems and Problem Solving in Technology.* Zeitschrift fr allgemeine Wissenschaftstheorie Vol.20, No.2, pp.212-229, 1989. For comparison see J.M. Bishop: *Computer Programming — Is is Computer Science?* Suid-Afrikaanse Tydskrif vir Wetenskap Vol.87, pp.22-33, 1991.

[9] Remember the similar debate concerning computer science and artificial intelligence some years ago.

[10] Confucius: *Conversations*, Chapter V, Verse 9.

[11] Confucius: *Conversations*, Chapter VII, Verse 8.