The pragmatic engineer versus the scientific designer.

The two literary figures mentioned in the above title were invented last week when I addressed young Burroughs managers on the role of formal techniques in automatic computing. (To sweeten the pill I presented them as "caricatures" although --but that is strictly between you and me!-- reality can only be understood as a gross exaggeration of fiction.) It was, I think, this recent exposure that, upon my return, made me read all sorts of planning and management documents that happened to cross my way --none from Burroughs, I am happy to add, because I was not very pleased with them-- with unusual attention. (Proceedings from a seminar on Digital Systems Design, a USAF survey on Software Engineering Project Management, and the minutes of a large number of meetings devoted to the planning of software research and development from several industrial organizations.) Among all these documents I suddenly saw a common theme.

* * *

I introduced the pragmatic engineer and the scientific designer as the two extremes of a whole spectrum in between. The pragmatic engineer believes in the correctness of his design until it has failed to work properly; the scientific designer believes in the correctness of his design because he has understood why it must work properly. And in order to drive home the message and its significance I introduced as exemplary thinking pattern of the pragmatic engineer the standard example known as "Poor Man's Induction": the "proof" that 60 can be divided by all smaller natural numbers: you just try! 1? Yes. 2? Yes. 3? Yes. 4? Yes. 5? Yes! 6? Yes!! OK.... Let us try a random example. 10? Yes!!! 12? Yes!!!!!!! Obviously 60 can be divided by all smaller natural numbers. After having done so, I continued my experiments both in public and in private: I used two very different examples, but in both cases it turned out to be impossible not to seduce my audience to the assertion that something was "impossible", whereas each time this assertion was solely based on the circumstance that the man making the assertion had been unable to discover a way in which "to do it". No discovery of a counter-example, was taken as proof. (Because my audience existed predominantly of people with their primary professional background in hardware design and development, and only too familiar with the extremely experimental attitude of

the average electrotechnical engineer in several European countries, including my own, I was not surprised at all by the outcome of my little experiment.)

Upon my return I gathered

1) that in hardware development it is still --I am writing November 1978-- quite common that a so-called "breadboard" is taken as the functional specification of the thing to be eventually produced in series --with the hilarious refinement that, because the breadboard might break down, in the case of an "important" design, two breadboards are made for the sake of "reliability": it makes one wonder what to do when the two breadboards turn out to differ functionally. Three breadboards and majority voting?--

2) that in software development the role of functional specifications is not properly understood. (I quote: "On a scale of 1 to 7, with 1 being little more than the name of the system, and 7 being complete specifications down through preliminary design, how detailed were the specifications provided?". From the way in which this question has been formulated one can deduce that it can only make sense --if it can do so at all: rating something along a scale from 1 through 7 in the absence of any metrics I still regard as silly-- within an environment that has not discovered yet that the main function of specifications is to act as a logical firewall between usage and design: if the functional specifications are "overcomplete" to the extent that they include "preliminary design" the firewall "leaks".)

3) that in both hardware and software development the role of documentation is not properly understood; it was uniformly treated as an additional burden, to be produced --by a special type of slaves called "technical writers"?-- a posteriori, therefore usually on the critical path and hard to keep up to date

4) that software project managers see software development primarily as a management problem, because they cannot manage it because they don't understand it --it was a very lively illustration of the saying that to someone, whose only tool is a hammer, every problem looks like a nail!-- (this week's sources gave no indication on the average hardware project manager's prejudices, but one almost feels entitled to extend one's doubts) .

The above list is impressive and --but this again between you and me!-- rather depressing.

* * *

Several people have told me that my inability to suffer fools gladly is one of my main weaknesses, and I believe them, but despite that belief I found so much wide-spread and persistent stupidity hard to understand. In order to ease my own amazement I came up with the following "explanation".

A main nigger in the woodpile is the invention --in Europe-- and the subsequent proliferation --primarily in the USA-- of the term "software engineering". The existence of the mere term has been the base of a number of extremely shallow --and false-- analogies, which just confuse the issue. In a recent issue of "Daedalus" I read an article by an American sociologist, in which he gave the popular picture of "the engineer". Parochial as sociologists usually are, he gave a very American picture; for me that was in a way illuminating, since, besides the similarities, he also gave me the differences. My conclusion was that the term "software engineering" should never have been coined. The typical engineer is an a-cultural illiterate, unable to absorb or appreciate carefully written prose, equally unable to express himself well, a socially deficient bore, whose primary role in life is to make new gadgets with his hands. (Particularly the "with-his-hands" part is still very dominant --perhaps even more so than in the USA-- in the older European Engineering Departments. Although..... Some time ago a Californian colleague wrote in a technical report that he was tired of proving the correctness of theorems "manually"! Being a little bit old-fashioned, I still use my brains for understanding.)

Looking at computers qua structure, qua challenge, qua potential, I can only conclude that the automatic electronic computer is a gadget absolutely without precedent. If that view is appropriate --and I believe it is-- we should be very suspicious of anything imported from elsewhere into computing --in the widest sense of the word-- with the mere justification that elsewhere it worked or made sense. Computers are such exceptional gadgets that there is good reason to assume that most analogies with other disciplines are too shallow to be of any positive value, are even so shallow that they are only confusing. I can only conclude that the wide-spread adoption of the term "software engineering" is to be regretted as it only hampers this recognition.

Let me give you just one example illustrating how serious the conse-

quences of the thus engendered confusion may be. One of the planning documents for software research revealed --in a parenthetical remark only-- an unchallenged tacit assumption by referring to "the tradeoff between cost and quality". Now in all sorts of mechanical engineering it may make sense to talk about "the tradeoff between cost and quality", in software development this is absolute nonsense, because poor quality is the major contributor to the soaring costs of software development. What can you expect from a planning document that is (implicitly) based on such a profound misunderstanding?

The wholesale adoption of the term "software engineering" may also explain one of the more surprising answers --it surprised at least me-- to the questionnaire that showed that the difficulty of getting adequate specifications was overwhelmingly regarded, not as a technical problem, but as a management problem! Now this surpised me, because even if the formal tools for giving the specifications are in principle available, how to apply them without the specification becoming unwieldy is a --technical-- problem whose solution is not apparent; in the absence of a ready-made formalism we return to our native tongues and we are almost back in the stage of Euclid who had to do mathematics verbally. (It is for this reason that an exceptional mastery of the native tongue of his environment is one of the programmer's most important assets.) It has even been argued that an important cause of the poor state of the art of programming is our collective inability of adequately presenting complex algorithms to our fellows --an opinion that is strongly supported by the ghastly quality of many publications-- . These are serious technical problems. Yet it is primarily regarded as a management problem! Can I offer an explanation by suggesting that those who answered to the questionnaire have tacitly assumed --or almost postulated-- that the technicians they have to work with, being "engineers" of some sort, are illiterate? If that were correct, how do they ever hope to "manage" around that incompetence? If I understand anything about anything, such a hope can only be based (again) on false analogies.

8 November 1978

Plataanstraat 5
5671 AL NUENEN
The Netherlands

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow