1980?

Jean-Raymond.

I can't remember when I wrote this. please return, I may use it some day.

Tony,

Abstract.

This paper argues that our recent progress in the development of a sound programming methodology should not lead us to ignore the more difficult aspects of engineering; and that in future we should pay more attention to the quality of our designs, ~~than merely~~ and not just the accuracy of their implementation.

Even in 1828 the profession of engineering had a long and honourable history; and since then it has made even more spectacular progress. I sometimes wonder whether we are yet justified in appropriating the title of Engineering to the profession of software construction; for its history to date is neither long nor conspicuously honourable. There is a danger that the use of an inappropriate title will not only debase the title, but will give us a dangerously false sense of achievements. I have previously tried to draw a comparison between the computer professional and the engineer of the present day; and this can be read in a book entitled "Software Engineering" published by Academic Press in the APIC series. I do not wish to repeat my arguments; but I must report ~~regret~~ that my conclusions were distinctly unfavourable to the computer professional.

So I think it is grossly unrealistic, not to say presumptuous, to begin to compare ourselves with engineers of the present day, whose "discipline" is based on a hundred years of mathematical and scientific research, and, on an (even) longer period of practical experience, including many failures, and many disasters.
Rather, we should look back to the time before the emergence of engineering methods, ~~when~~ to the age of the master craftsman. The engineering achievements of these master craftsmen were quite remarkable; they were responsible for many excellent ships, sturdy and elegant furniture, (sound bridges) and spectacular cathedrals, (unmatched in more recent times.) With the materials and tools then available, no engineer could have achieved more.

So it is not primarily in his achievements that an engineer is distinguished from the craftsman, but rather in his tools and in his methods. The craftsman knows what he he is going to build, and knows how to build it. He has no need of elaborate plans, scaled blueprints, careful measurements, precise quantities, progress charts, delivery schedules and cost estimates. When he undertakes to make something, he succeeds, because he knows how to make it (and his customer knows what to expect). If by chance something goes awry, he knows how to adapt his work or his design to compensate for the error. And in the end, his product works, and gives good service, and endures. Or, on the other hand, it doesn't! In that case, it merely shows that the craftsman was not such a master as he thought he was; and he would not get the next commission. And so, over a long period, by a process akin to natural selection, only the fittest craftsmen and the fittest designs will survive.

He seems to have an ingrained sympathy with his materials, and an intuitive knack for handling his tools most effectively.

And so it is with the programmer, systems analyst, or project leader of the present day. He may indeed start with a description of what his client thinks he wants; but the description is so imprecise, inconsistent, and even inconstant that it can serve only as a rough diagram rather than ~~as~~ a firm plan for implementation. ~~But~~ Nevertheless, a good programmer knows how to proceed. He seems to have an intuitive grasp of his programming language and an improved feeling for what his operating system can be made to do. He starts writing and testing his code, and when it is all finished, it all (miraculously) fits together and works after its fashion. If anything goes awry, he hacks a bit at his already written code, modifies his plans a bit, and after some delay, delivers his product. If it isn't quite what his client wanted, he can continue to hack until the client is satisfied; r, more usually, until he gets tired of waiting.* And if the product never gets to work at all, or is too inefficient or expensive to put into use, there is no point in trying to understand why — ~~why there is no underlying~~ it is just one of those things that happen in programming. So we cannot even learn from our mistakes!

* In a well-documented case, this has required over twenty major reissues.

Perhaps the engineer has the advantage that when his products fail, someone gets killed: so that a competent formal enquiry is set up to probe ~~it~~ the cause of the disaster, and if necessary to apportion the blame to the engineers responsible.

The method of training of the craftsman is also distinctive. No formal instruction in reading, writing, or arithmetic is required. A young lad would be apprenticed for a number of years to a master, and serve as his drudge, and assistant, and ~~probably~~ whipping boy; and in return, he would have the privelege of watching the master at work. ~~And~~ At the end of a satisfactory apprenticeship, ~~the apprentice he might~~ he might be worthy of employment as a paid assistant; after an even longer time, he might be good enough to set up as a craftsman on his own, and hand on the craft to a new generation of apprentices.

new para

Is this not rather similar to our methods of training programmers and systems analysts? The raw recruit has been well drilled by our educational system that reading is irrelevant, ~~boring~~, writing difficult, and mathematics impossible. After a few weeks acquaintance with the esoteric mysteries of some standard programming language, he is thrust into a team engaged in some half-finished project.

Some small and unimportant part of the project is allocated to him. When the project is complete, (or even before) the experienced members of the team go off to start a new project, and he is left behind on "care and maintenance" duties. If he is lucky as well as wise, he will learn by study of the program for which he is responsible that ~~the value of~~ sound methods of program specification, design, coding, and documentation are both possible and necessary. But just as likely he will learn that programs are hastily contrived, carelessly botched together, full of errors, and unexplained and inexplicable, neither useful nor convenient to man nor beast.

It is now beyond time to turn to the transition between a craft and an engineering discipline. How shall we characterise its first steps? I would like to suggest that the first engineer was the man who designed and planned his product on paper, and measured his designs with sufficient accuracy that they could be implemented by competent artisans without his personal supervision; and when the separately implemented components are brought together, they mesh exactly as he had intended.* The formal technique which underlies this remarkable achievement is the scaled diagram, which enables a complete design to be specified in as much detail as desired, and enables the consistency of the design to be checked before the smallest part of it ~~was~~ is implemented. As an extra bonus, it makes possible an accurate advance estimate of the characteristics of the product — its size, and weight; and of the amount of materials and number of components, and length of time required in its manufacture.

---

* In a well-documented case, the fitting together of separately implemented program components actually consumes more effort than their separate implementation.

Of course, the use of these new techniques requires new insights and new skills: the use of rulers, compasses, set squares, parallel bars) (and protractors; a facility for meticulous line drawing, and even a familiarity with elementary arithmetic and geometry, which provide~~s~~ a modest theoretical foundation (for the technique.) What scorn these newfangled methods must have evoked from the master craftsmen,* especially when they were applied inconsiderately to bad designs of unsuitable products. But the advantages of the new method~~s~~ was so great that they ~~it was~~ now ~~were~~ universally adopted; ~~and nowadays~~ teams of professional draughtsmen and checkers have a recognised and essential role in every engineering project; and their methods are enshrined in highly developed " standard codes of practice; and their skills are taught at technical schools by generations of teachers to generations of students.

** Even today, there is a powerful call for a return to the age of the master craftsman — more fashionably known as a chief programmer.

Perhaps our first faltering steps towards a discipline of software engineering are rather analogous. They are based on the discovery that a program can actually be designed before it is written, just as a table can be designed before it is constructed; and that the design can be written down on a piece of paper with sufficient accuracy and rigour that the various components can be implemented by others, and can then be correctly assembled into a working product. Of course, the design is not pictorial; it is a symbolic representation of the preconditions and postconditions of the program as a whole and of each part of it. But the advantages of these logical assertions is the same as that of engineering and architectural drawings: they enable the consistency of the whole design to be checked before any part of it has been implemented.

And there is the same additional bonus.
The scale and complexity of the logical assertions give some advance indication of the size and efficiency of the product, and the cost (and timescales) of its implementation.

Of course, the use of these new techniques requires new insights and new skills: reasoning in the first order predicate calculus, and an understanding of the elementary properties of numbers, sequences, sets, and mappings. What scorn these newfangled methods evoke from the master programmers! But I would suggest that within the next few generations of programmers they will be universally adopted for program design and documentation; and methods of ~~implementing~~ transcribing these designs accurately in COBOL or FORTRAN will become highly standardised and widely practiced. ~~and these skills will be taught at technical schools to generations of students~~

The widespread adoption ~~development~~ of a sound methodology is the first and most necessary stage in the development of a proper engineering discipline. We have made a good start in revealing the logical basis of a programming methodology, and we are beginning to test and develop it on more significant problems. But even when we have achieved to the full our research aspirations, — when the activity of programming has been elevated to a routinely accurate procedure, widely taught and universally applied — even then we shall not have earned the title of engineers. There is more to engineering than merely standard drawing office practices.

I think that the mistake which we risk making is that we pay far too much attention to the methods and techniques of implementation, and far too little to the quality of the design being implemented. It is possible to use the most refined and accurate methods to implement the most inadequate designs. There is nothing wrong in the drawing office procedures used to construct ships that will hardly stay afloat, planes that hardly fly, schools which act as solar heat traps, and roads which cause traffic congestion. Similarly, we can imagine the flawless application of the most advanced programming methodologies to a sorting program that sorts in n factorial time, a programming language with a manual of two hundred pages, a compiler that take ten seconds to compile the null program, and operating systems that occupy three or four megabytes of storage to do even less. I fear that for many of us, this is no mere imagining

What is wrong with these products is not the skill with which they are put together — that one can only admire! It is just the grotesque inadequacy of the original designs, which have emerged by an apparently random historical and political process, and pay not the slightest regard to the ~~old ideals of the engineer, the direction of the great power of the electronic digital computer to the use and convenience of man.~~ most elementary concern for matching technique to objective and objective (to technique,) for minimising cost and maximising benefit, in short for serving the use and convenience of man.

So I would like to suggest that if we are to deserve the title of software engineers, we must pay a great deal more attention to the quality of our designs. This is very much more difficult to learn and to teach than the practice of sound ~~implementation~~ programming methodology. It requires ~~diff~~ a deep understanding of true and reasonable requirements of the intended user of our product. Not only must we have a better understanding than its user — we must persuade the user of this fact. ~~which~~ In the present day this can be more difficult; - and, let's be honest, we do not yet deserve our clients' ~~lack of~~ confidence.

We shall also need a very wide acquaintance with the computer techniques which are available to meet our clients' needs; and the good judgement to select the technique which gives best value for money. At all stages, each design decision must be taken with a clear knowledge of the consequences of that decision, both on other aspects of the design and on the quality of the final product. Above all, the good engineer must never allow the complexity of his design to exceed what he knows that he can comprehend and control, not only when all is going well, but also when things begin to go wrong.
In every branch of engineering such a degree of competence (and modesty) is now taken for granted. In ours, we can hardly yet make such a claim.

There is another reason why engineering is so much more difficult than mere methodology: and that is, that there is no single unified discipline of engineering.

~~So my final suggestion is that we should recognise that perhaps there is no single unified discipline of software engineering, just as there is no single unified discipline of engineering.~~ We have civil engineers, architects, electronic engineers, naval architects, electrical engineers, mechanical engineers, (aeronautical engineers,) production engineers, etc. All their training and knowledge and experience is specialised to a particular area of application; and if required to undertake a task outside that area they would be as incompetent as the rest of us. It is only at the lowest and most trivial levels that there is anything in common between these branches of engineering — ~~I refer to the level of the accurately scaled drawing.~~ That is, at the level of standard drawing office practice.

In just the same way, I would suggest that we should recognise that there is no single unified discipline of Software Engineering.

~~If we are ever to reach our engineering ideals, we must learn new skills not only in implementation but also in design. And here I do not believe that there is any single discipline which will be suitable for all areas of application.~~

same para

To design a good sorting program one must know about sorting; to design a good numerical algorithm one must know about error analysis; to design a good compiler, one must know a lot about compiling techniques; to design a good programming language, one must know a lot about compilers and even more about programming. And none of these skills would give the slightest qualification for the design of an operating system. It is only at the lowest level of methodology that these many specialities share a common code of practice.

In this keynote address, I have tried to suggest that we are still a long way from deserving the title of Software Engineers. Let me end on a more constructive note. I wish to address the question how are we to train good engineers in all these specialised areas. The answer is taken from other disciplines, where the art of design can be assimilated by studying a wide variety of excellent designs produced by the ~~brilliant~~ best engineers of the past. ~~It is mainly~~ But this presupposes that these designs are available in a form sufficiently complete and precise to be a suitable object of study. ~~Up till Hitherto, we have had no gen suitable method of generally accepted frams notations for conven presenting such~~ Hitherto, the study of other programmer's designs has not been a pleasant duty; and this may be due largely to the grotesquely inadequate methods for precise specification of these designs. But this problem is being solved by the rapid progress of our study of programming methodology, which permits a rigorous description of program designs in a form which can be studied both for illumination and for enjoyment.

In the later sixties these were superseded by software of very much more complicated and inferior design

But where is the wide variety of excellent designs produced by the best engineers of the past? There are many examples from ~~remembered~~ the early nineteen sixties — ~~compilers~~ for example, compilers and operating systems that were so good that their users never found anything to complain about them. Many of these were ~~written~~ constructed by extremely primitive methods, using assembly code or even machine code. It was only by the quality of their design that they commended themselves. I propose that these early designs should be recreated using modern design methods and descriptive techniques, and should serve not only as ~~a~~ models and instructional material in the new methodology, but should also convey a good understanding of the factors and techniques which contributed to the quality of the design, and above all, the meticulous attention paid by the original designers to the practice of directing the great computational power of the electronic computer to the use and convenience of man.