

# Is Software Engineering Really Engineering?

Ray Offen

CSIRO-Macquarie University Joint Research Centre for Advanced Systems Engineering  
North Ryde, NSW 2109, Australia.

## The Problem?

From my perspective, this somewhat ill-posed question misses the point rather badly, because as Michael Jackson stated correctly and unambiguously in his ICSE-17 keynote address, "as software developers *we are* [my italics] engineers because we make useful machines ..." [1]. This is self-evident if one accepts that most conventional definitions of *engineering* refer to creating cost-effective solutions to practical problems by applying scientific knowledge to build things in the service of mankind, or some words with the same overall meaning [2].

For someone who has spent much of his career, as both scientist and engineer, building large systems or researching their development, the real question is how we *improve* the engineering activities that we undertake as software engineers. This is important, because in our daily lives increasingly we depend, often for our very lives and livelihoods, on the 'correct' execution of software in a machine. We are intimately concerned with issues of functionality, reliability, performance, safety, security and a raft of other concerns, all mediated by software.

In practice, it is certainly true that the actual *engineering* component of software engineering is at times very fragile. This is quite different from stating categorically that this is *not* engineering! Shaw [2] has described how engineering only emerges once commerce-based *craft* [skilled craftsmen involved in pragmatic refinement] and *science* converge to give a true *engineering* discipline [educated professionals, using science-based analysis and theory]. This convergence is still in the process of taking place for the building of software intensive machines - hence the ambivalent views of software engineering that now exist, even among professionals. On the one hand software flies planes, controls industrial plants, mediates car braking systems, manages hospital intensive care units and so on. On the other hand, some US states have legislated that use of the term "software engineer" is expressly forbidden. Closer to home, this APSEC '95 conference is posing the question "is Software Engineering really Engineering?", with the implication that it is not! A schizophrenic view of software development indeed!

What are we actually talking about here? Computation is a man-made phenomenon, focused primarily on abstract intellectual artefacts, frequently involving great complexity, underconstrained by its medium and only weakly coupled to theory. In fact, Computer Science has assembled and developed some relevant theory, but practice proceeds largely independently of this organised knowledge. The weak link between theory and practice is exacerbated by a poor understanding of experimental methods in the Computer Science research community and the fact that 'theory' is closely, and wrongly, associated more with the mathematical sense of the word. In Computer Science standards for demonstrating correctness are very close to those traditionally used in mathematics, where theoreticians 'prove theorems', compared with the building of constructive analytic models which is central to theoretical thinking in other sciences. An obvious consequence of all of this is that models and modelling become crucially important, in that they alone can give substance to the abstract intellectual artefacts characteristic of both software product and process and enable them to be subjected to shared comprehension and scrutiny. Indeed, it can be argued that many of the problems that beset software engineering are in fact problems with modelling. For example, both reuse and codification require shared and understandable models if they are ever to be effectively utilised. Codification is a vital ingredient of any engineering culture and a lack of suitable, widely understood, tool based modelling frameworks mitigates strongly against effective codification and the effective use of already codified experience. Effectively capturing context is a crucial issue in a world of abstract artefacts, especially in areas such as estimation and process improvement.

## The Solution?

So, what is to be done? Here is a personal view of some of what still needs to be undertaken. Fundamentally, there is a need to provide appropriate science and technology for speeding up the craft to engineering transition. This is not necessarily easy: in other engineering disciplines this process has taken centuries! The centrality of software in our

technology dependent society predicates a speedier transition, hopefully guided by our insights into the process as exemplified in other engineering disciplines.

As was prefaced above, a proper understanding of the role of models and modelling in software engineering is crucial, especially with respect to 'deep' analytic models. This is an essential precursor to effective codification - ie experience *reuse* - and the development of CASE tools that actually provide the sorts of design and development support designers take for granted in the mechanical and electronic engineering CAD world. An obvious facility would be the ability to *animate* an emerging system design early in the software design process.

Two further related and important areas of research are the use of quantitative data [software measures] for the building of more powerful predictive models and empirical software engineering studies. These are intrinsically difficult areas of research, both because of the abstract nature of many of the intellectual artefacts which constitute software products and the associated development processes and because of the human/social-centredness of the software development process. The net result is that conventional approaches to model building and experimentation, as found in the physical and social sciences often do not carry over naturally into the software engineering domain - a combined socio-technical viewpoint must be pursued. A further complication is the very considerable cost associated with conducting meaningful software engineering experiments in realistic industrial contexts. A major contributing area of statistical research is the difficult topic of meta-analysis, previously mainly used for the analysis of multiple, but simple, medical experimentation datasets, but now being extended so as to cope with complex hierarchical models (such as a software product, development process or project organisation). Progress in this area is vital if intrinsically sparse experimental software engineering data-sets are to be combined effectively in such a manner as to realise generalisable predictive models - the key to codification

There is much more that could be said, but as an aside, there some promising signs that the as yet small science base is starting to grow. For example, the recent use of higher-dimensional models, making novel use of category theoretic 'pasting schemes', for modelling important aspects of concurrency [3]. Computer Scientists tend to have a strange aversion to higher-dimensional models, very much part and parcel of traditional science. The tacit assumption that most software complexity measures are scalars, as opposed to vectors or tensors in some 'space', is

a glaring example of very limited conceptual thinking and an inability to construct meaningful empirical models of the relevant context. Another positive example is recent work in belief revision and reasoning about action which avoids many of the drawbacks characteristic of the 'frame problem' in AI by making use of fundamental invariants in reasoning about system behaviour [4]. In the physical sciences, invariants and their concomitant symmetries play an important role in building powerful analytic models of physical systems. There is a real need for more work with a core science focus and of this quality.

The education and training of computing professionals in general, and software engineers in particular, is a central issue. These must be based on sound science and engineering principles with plenty of relevant practical experience - sadly not the case today in most Computer Science departments, where the academic staff 'teaching' software engineering frequently have no experience whatever of the reality of building high-quality, reliable, cost-effective software systems. One thing we can be sure of is that *certification* of practicing software engineers, through the engineering institutions, will come whether we like it or not! There is nothing so special about software engineering, compared with other branches of the engineering profession that makes us immune from such professional constraints - and benefits. We should welcome it and work constructively towards it. Then we really will be engineers, irrespective of what we choose to call ourselves.

## References

- [1] M. Jackson, "The World and the Machine", *Proc. 17th Int. Conf. on Software Engineering ICSE-17*, Seattle, Wa., USA, April 23-30, pp. 283-292, 1995.
- [2] M. Shaw, "Prospects for an Engineering Discipline of Software", *IEEE Software*, vol. 7, no. 6, pp. 15-24, November 1990.
- [3] R. Buckland, "Choice as a First-Class Citizen", to appear in *Proc. ISLIP '95*.
- [4] P. Peppas, "Belief Change and Reasoning about Action", PhD thesis, University of Sydney, Australia, December 1993.

**Ray Offen** BSc PhD CEng FIEE FIREE is the Director of the CSIRO-Macquarie University Joint Research Centre for Advanced Systems Engineering. He also holds the Chair of Information Technology at Macquarie University and is Chief Research Scientist in the CSIRO Division of Mathematics and Statistics. Prior to this, he has held a number of senior industry and academic posts in science and engineering.