

Software Engineering Education - is it meeting industry needs? Can industry needs be met?

(Panel Session)

Panel Coordinator - Val Veraart
CSIRO Tropical Agriculture
Brisbane, Queensland
Val.Veraart@tag.csiro.au

Mr Martin Hilton
Software Resources Manager
Motorola Australia
Adelaide, South Australia
mhilton@asc.corp.mot.com

Associate Professor Doug Grant
Head, School of Information Technology
Swinburne University of Technology
Hawthorn, Victoria, Australia 3122
doug@saturn.csse.swin.edu.au

Mr N. Samy
Software Engineer
GEC-Marconi
Sydney, New South Wales
LNSAMY@gecms.com.au

Mr Tony Greening
Associate Lecturer
Basser Department of Computer Science
University of Sydney, New South Wales
tony@cs.usyd.edu.au

Introduction

For a number of years a mismatch has existed between the skills that industry, in general, deems essential and the knowledge that tertiary institutions instil into their graduates. Until recently this mismatch has been evidenced by, on one hand, many employers requiring graduates to possess skills in specific languages and methodologies that ensure that they can "hit the ground running" when they gain employment whilst the Universities response, in general, has been to express the view that their role is to teach the concepts and building blocks of their discipline rather than expertise in any particular language.

The mismatch appears to be worsening as two new factors become evident. The first of these is economic. Many employers can no longer afford to put in place extensive training programs for new graduates to gain expertise in specific applications, languages and methodologies. The second, and major factor, is the explosion of skills that define the discipline of software engineering. For example, in recent years an emphasis on team work, human interactions and 'process' skills

such as estimating, scheduling, configuration managements and metrics has been steadily emerging. Add these to issues of networking, the world-wide web, data warehousing and fault-tolerant systems and we have a veritable explosion of information technology generally. Can education cope? Is industry reasonable in its expectations?

Some issues for exploration include:

- is our industry's requirements-list for graduates realistic?
- is the University position tenable as pressures on cost-effective education increase?
- should a software engineer need such a wide range of skills, or is it time to differentiate into some para-professional careers, for which there are precedents in engineering disciplines?
- should the Universities seek homogeneity in their degree programs or should they differentiate themselves into 'areas of excellence' at undergraduate level?
- is it possible to teach 'team' and 'human-interaction' skills to undergraduates, or can

they only be appreciated after some time in the workforce, and

- do we need specialised graduate Masters course, and will they get support from employers and employees?

Position Papers received at time of going to press

Doug Grant

In a three or four year undergraduate program, students gain the foundations for their professional careers. They cannot be expected to become expert. They cannot be expected to gain professional level skills in a very wide variety of development methodologies, tools or application areas. They cannot be expected to acquire the ability to achieve the productivity levels of experienced software engineers.

What can be achieved is an education, in partnership with industry, that recognises the initiatory function of a degree, and its fundamental requirement of providing preparation for a lifetime of productive work and lifelong learning.

World-class software organisations recognise the need to provide product and methodology training to new graduates upon commencement with the company. (My recent discussions with a number of the most successful companies in the booming Indian software industry stands as testimony to this statement.) World-class universities prepare their graduates to be effective learners when taking such courses. My position is that universities should embrace the principle suggested by these points - that laying foundations for professional life is their fundamental role, not to be compromised by short-term needs of some organisations unwilling to accept their on-going responsibility for a partnership with their employees in lifelong learning and training.

In the undergraduate software engineering curriculum, emphasising the 'soft' skills of communication and teamwork, and ensuring that students gain a solid appreciation of organisational and business perspectives, is essential. So are presenting the disciplinary foundations in both technical and managerial domains that most professional software engineering educators and practitioners agree on in principle, if not in detail. Enabling students to gain the fundamental problem-solving skills of abstraction and concretisation is vital.

The temptation to crowd the curriculum with too many of the latest and greatest technical innovations can and should be avoided. Historically, surveys of industrial

employers have regularly indicated that the imperative in undergraduate education does not include extensive, detailed exposure to the complete range of contemporary developments; usually a core set of such developments can be identified that suit most employers. From time to time, these requirements change.

I am personally persuaded that by forging close links between academia and industry, and using industry advisory committees with significant involvement in curriculum development, we can achieve undergraduate curricula which meet general industry needs. The success of such undergraduate programs is enhanced by embracing cooperative education principles, which involve students spending a considerable time (preferably at least a year) during their degree program working in industry.

Tony Greening (co-authors Judy Kay and Jeff Kingston)

Problem-based learning in foundation computer science: a software engineering orientation to the discipline.

Our department has made radical changes to the way we teach our foundation courses in computer science. The main elements of the changes are:

- teaching using an approach called Problem-Based Learning
- Blue, an object-oriented programming language that has been designed and implemented at Basser
- sections breaking the large class (about 700) into smaller section (each with about 100 students)
- streams enabling students with differing
- interests to study their computer science in terms of different problems.

These changes are timely. They are intended to address the particular problems of the changes on technology with a move to object-oriented programming and a focus on co-operative group work as the norm in many workplaces that Information Technology specialists work.

At the same time, they recognise the many problems that have been identified as damaging the performance of students in their first year at university: a sense of isolation, lack of affiliation to their course and loss of a sense of the purpose of their study. These problems are especially disabling for students in large courses.

Although the change in programming language and paradigm is important and makes for significant differences in the way that students need to learn, the shift to problem based learning is probably a larger shift