Chapter VIII
Aligning Learning with Industry Requirements

Jocelyn Armarego
Murdoch University, Australia

ABSTRACT

A review of studies of practitioners of software development reveals a depth of mismatch between their needs and formal education. The conclusion to be drawn is that industry has made a long-term shift in its requirements of graduates from technical subjects, laying emphasis on personal and affective attributes. Concern has been expressed that the underlying "socialisation" requirement for a graduate to achieve "working professional" status is very poorly addressed in formal education. After establishing a framework for comparison between information technology (IT) formal education and industry requirements, this chapter discusses an action research study based on applying nontraditional and innovative learning models to address mismatches identified. Results suggest that models which focus on independent learning and soft skills prepare students to enter industry with the ability to engage in the career-long, professional learning required for success in professional practice.

INTRODUCTION

Software development has been described as a "craft." The negative connotations of this label include an inability to consistently guarantee a quality product, fit for the purpose for which it was developed, produced on time, and within budget. As an example, a mid-1990s study of over 8,000 projects (Standish, 1995) indicates only 16.2% of software was successful. These rates do not significantly differ from those reported in the 1970s and 1980s (Mann, 1996). The issues that underlie this state-of-affairs (namely, intrinsic difficulty, uniqueness of each system, multidisciplinary skills necessary, and a requirement for life-long learning in practitioners) are described later on.

A review of major model curricula for software development (e.g., information systems [IS], computer science [CS], and software engineering [SE]) shows that, in general terms, a graduate within the broad IT discipline should emerge from formal education with knowledge of the basic software development processes (and therefore, in theory, be able to produce successful software). While practitioner studies indicate that the base case of content knowledge is covered in models used in university programmes, a closer look reveals the depth of the mismatch between practitioner needs and formal education in software development in general.

Engineering Software

Those involved in the development of software agree that one mechanism for dealing with the intrinsic difficulties (e.g., complexity, visibility, and changeability [Brooks, 1986]) of developing successful software was to embed its production within an applied science environment. Royce (1970) was the first to note explicitly that an engineering approach was required, in the expectation that adhering to a defined, repeatable process would enhance software quality.

This interest in engineering is mirrored in the education of software developers, with an exponential growth in offerings of undergraduate software degrees within an engineering environment. Increasingly, this education focuses on process and repeatability, modelling scientific and engineering methodologies. The underlying assumption of this approach is that "good" software development is achieved by applying scientific investigative techniques (Pfleeger, 1999).

Creating Software

There are positive implications as well for the label "craft." Each system is considered a unique synergy between the hardware, software, and organisational context in which it will be utilised. This approach suggests that the development process cannot be repeatable, as the forces at play will differ for each context; continually changing as understanding of the characteristics of the developing system grows in all stakeholders.

From this perspective software is a collaborative invention. Its development is an exploratory and self-correcting dialogue (Bach, 1999), based on insight-driven knowledge discovery (Guindon, 1989) facilitated by opportunistic behaviour (Guindon, 1990; Visser, 1992).

The risk is that strict adherence to engineering and science methodologies hampers the quintessential *creativity* of this process (Lubars, Potts, & Richer, 1993; Maiden & Gizikis, 2001; Maiden & Sutcliffe, 1992; Thomas, Lee, & Danis, 2002). These, potentially:

- Restrict essential characteristics such as opportunism (Guindon, 1989)
- Assist in adding accidental complexity through their attempts to control professional practice (by restricting natural problem solving, Sutcliffe & Maiden, 1992)
- Impose a plan at odds to inherent cognitive planning mechanisms and hence interfering with the management of knowledge (Visser & Hoc (1990) suggest that, in practice, a plan is followed only as long as it is cognitively cost-effective)

Practicing Software

The skills and knowledge required to be active as competent professionals are multidisciplinary. For software development, Zucconi (1995) suggested the underlying disciplines of central importance are psychology, CS, and discrete mathematics, and suggests an IT professional needs to be well organised, able to work as a member of a multidisciplinary team, and able to work within the scope of the employer's policies and procedures and society's tenets.

This equates well with the stated needs of practitioners. Practitioner-based studies (Lee, 2004; Lethbridge, 2000; Trauth, Farwell, & Lee, 1993) and in the Australian context (Scott & Yates, 2002; Snoke & Underwood, 1999; Turner & Lowry, 2003) assist us in building a profile of a practicing IT professional.

Related Content