Chapter 16
Lessons from Practices and Standards in Safety-Critical and Regulated Sectors

William G. Tuohey
Dublin City University, Ireland

ABSTRACT

Many years of effort have been expended by experienced practitioners and academic experts in developing software engineering standards. Organizations should see it as a positive advantage—rather than as a costly negative necessity—when they are required to develop software to a recognized standard. A genuine, constructive program of measures to ensure compliance with an objective standard will achieve development process improvements that would otherwise be difficult to motivate and bring to fruition. This chapter provides an overview and comparison of a number of software engineering standards specific to safety-critical and regulated sectors. It goes on to describe implications and benefits that flow from these standards. Informed by current software engineering research, suggestions are made for effective practical application of the standards, both at individual project and at organizational level.

1. INTRODUCTION

Ten years ago, in a retrospective on data for 12,000 projects, Jones (2003) found that defect removal efficiency level was highest for "systems" software including safety-critical software, attributed in part to the use of reviews, inspections and intensive test activities; overall, good quality control was found to be the best indicator for project success. Since then, the world has become more and more dependent on software-intensive systems and there is a constant struggle to deal cost effectively with the great increase in system size and complexity while continuing to ensure safety (Larrucea, Combelles, & Favaro, 2013). Of course, most software is not safety-critical; indeed, "Nowadays, software is everywhere and is often built by relatively inexperienced programmers" (Abdulla & Leino, 2013) and "most programs today are written not by professional software developers" (Andrew et al., 2011). On the other hand, much software, including commercial applications, is built on top of middleware (Emmerich, Aoyama, & Sventek, 2008) which should certainly be well engineered.

DOI: 10.4018/978-1-4666-6026-7.ch016

This chapter is based on the belief that all software development can benefit from the experience of the safety-critical and similar communities, both in terms of successful past and present practices and in how future challenges are being tackled. The focus of the chapter is mainly on general practices and especially software engineering standards, but it is pointed out that there are many specific lessons to be learned also. For example, Durisic, Nilsson, Staron, and Hansson (2013) are concerned, for the automotive industry, with monitoring the impact on architecture of on-going changes but note that "…it is possible that the metrics are applicable to a wider range of software systems which rely on communication between different modules over multiplex buses". Apart from technical aspects, commercial benefits can be achieved by improving quality and, where applicable, regulatory practices in line with the criticality of the product (Meagher, Hashmi, & Tuohey, 2006).

There are very many published software engineering standards, some generic, some applicable within specific industrial sectors, some originating from particular procurement agencies, some developed by professional bodies, some relevant to certain categories of software (Tuohey, 2002). The multiplicity of standards, and the sometimes (necessarily) dry and legalistic style in which they are written, tend to make this material inaccessible to both software engineers and managers. Yet the standards may impose far-reaching constraints on day-to-day engineering work, on project procurement and management, and on an organization's commercial performance. This chapter presents a synthesis of a number of representative standards. This is achieved by first providing, with the aid of diagrams devised by the author, a somewhat detailed overview of the well known civil aviation standard RTCA/DO-178C (2011). It is hoped that this overview will be of practical utility to readers with particular interest in that standard. Next, more briefly, a selection of other standards is described in terms of their similarities and differences with respect to that reference. It is believed that RTCA/DO-178C (2011) is a good point of reference in that it is mature, is used by a wide variety of development organizations, and is applicable to software of different levels of criticality. Remarks on the evolution of this standard from its predecessor RTCA/DO-178B (1992), to which backward compatibility was intended, are included in section 3.1.

Software engineering standards, together with supporting guidance and related material, constitute a considerable resource for the software developer. Some indications are provided in the chapter of the nature of the available material. A goal of the chapter is to provide practical suggestions on how to apply software engineering standards effectively. A particular effort is made to identify project- and organizational-level characteristics that follow from or are supportive of such standards. A key consideration is that measures taken should be effective and efficient.

Very often, software development takes place within a wider system development context. This is made explicit in some of the standards, especially those dealing with safety-related systems such as RTCA/DO-178C (2011) which, in comparison with RTCA/DO-178B (1992) and drawing especially on ARP4754A/ED-79A (2010), includes a significant amount of new material on the overall system. Evidently, this wider context may impact on or constrain how a software development is organized. While this chapter is mainly focused on purely software issues, some of these "wider" impacts are noted. In this context it is interesting to note the remark in Mashkoor and Jacquot (2011) that "by contrast [with older engineering disciplines] in software engineering, systems are sometimes developed by people with an incomplete knowledge of their particular domain".

In the past, rigorous software engineering standards were imposed mainly on certain kinds of software development, particularly those with safety impact or with stringent reliability requirements or of a clearly mission-critical nature (such