A New Standard for Quality Requirements

Jørgen Bøegh, *Terma A/S*

A new standard on software quality requirements, ISO/IEC 25030, takes a systems perspective and suggests specifying requirements as measures and associated target values.

The ISO recently published ISO/IEC 25030, a new standard on software quality requirements[1] that complements the two IEEE Computer Society standards for software[2] and system[3] requirements. These three standards are important, given that properly identifying and specifying requirements are prime factors in determining a software project's success or failure.[4] Many companies, having realized this, are now better emphasizing requirements specification. Unfortunately, there's a tendency to focus on functional requirements rather than quality issues such as usability, maintainability, reliability, portability, and efficiency.

ISO/IEC 25030 can improve software quality by helping developers identify and specify quality requirements. Here, as an editor and a member of the ISO committee, I discuss some of the thoughts behind ISO/IEC 25030 and briefly summarize its main points.

Developing the standard

The ISO first decided that quality requirements deserve their own standard back in 2001. It then implemented the idea in connection with a planned revision and restructuring of two international standards—ISO/IEC 9126 (which presents a software quality model)[5] and ISO/IEC 14598 (which discusses software product evaluation).[6] The ISO/IEC JTC1 SC7 committee included the quality requirements standard in a new SQuaRE (*Software Product Quality Requirements and Evaluation*) series of standards, designated by the number 25000.[7] SQuaRE includes five divisions: quality management, quality model, quality measurement, quality requirements, and quality evaluation (see table 1). The quality requirements division contains the new ISO/IEC 25030 standard.

Taking a systems view of quality

When writing the standard, the committee members quickly realized that you can't elicit software quality requirements without taking a systems perspective. Software is normally part of a larger system, so you must view software requirements as part of the system requirements. We therefore looked at ways to describe systems. We wanted to develop a system model that was powerful enough to capture software quality requirements yet was easy to understand and still focused on quality.

A system is a combination of interacting elements organized to achieve one or more stated purposes.[6] The simplest system of possible interest when considering software quality is a computer system, which comprises three elements: hardware, software (including the operating system and application software), and data. This system model covers software running on a single, stand-alone computer and has been the implicit conceptual model behind software quality for many years.

However, the computer system isn't a realistic model. Software is often distributed on many computer systems—consider, for example, client-server systems and Internet applications. We needed to

0740-7459/08/$25.00 © 2008 IEEE