ThinkingISsues

Tony Clear
School of Computing and Mathematical Sciences
AUT University,
Private Bag 92006, Auckland 1142, New Zealand
Tony.Clear@aut.ac.nz

A 'Potted Guide' to Quality Assurance for Computing Capstone Projects

The topic of Quality Assurance (QA) is a key assessment category for the capstone project within our Bachelor of Computer and Information Sciences. In the past this has not been too much of a problem to explain to students, as they had typically studied several software development courses within which the concepts and practices of software quality assurance had been covered.
More recently we find ourselves explaining the concept of quality assurance to students from less software intensive majors (e.g. IT security), and finding it more of a challenge. Students are required to produce an evidence portfolio and QA is one of the categories under which they must demonstrate how the quality of their work has been "assured".
The initial questions arise of: what is quality? How can it be ensured? Then to start from scratch with defining quality raises further questions. In the literature for example, software quality has been termed "the elusive target" [9], which can be viewed from five different perspectives:

- The *transcendental view* sees quality as something that can be recognized but not defined.
- The *user view* sees quality as fitness for purpose.
- The *manufacturing view* sees quality as conformance to specification.
- The *product view* sees quality as tied to inherent characteristics of the product.
- The *value-based view* sees quality as dependent on the amount a customer is willing to pay for it.

Each of these perspectives brings accompanying approaches to ensure that quality outcomes result from the work performed. For instance '*transcendental*' quality might result in product solutions characterised by elegance of design, simplicity and ease of use, a crisp, clean look and feel, or a service marked by attention to detail and true attentiveness to the customer's needs. To determine the requirements in such a model of quality is challenging as they go beyond '*fitness for purpose*', to often *latent* and unstated needs and wants. Iterative lifecycle models with regular customer feedback loops may suit such projects. By contrast the '*conformance to specification*' model demands that a clear initial specification is produced and agreed, and the lifecycle model is normally typified by stage gates and customer sign-offs. Finally while the '*product view*' may place considerable emphasis on the design aspects of quality, the '*value based view*' in turn may be appropriate if a quick prototype is required for a one-off solution.
From a different viewpoint again, if we build the learning dimensions of the project into the equation there are differing models for educational quality. The one I prefer is that espoused in [6] whereby learning is conceived as a '*transformative process*' for the student – and therefore somewhat akin to the 'transcendental view' of quality.
As is apparent from the above, there are differing ways of producing quality outcomes depending upon the goals. From a student perspective the most useful way of framing this is to note the

need for a *quality process* to ensure predictable and high quality outcomes. This is where selection of a methodology to fit the needs of the project is necessary. That methodology might involve: a software lifecycle model; a technology or network design model (e.g. [12]); or a research design model (e.g. 11]). This choice demands making explicit whichever model of quality from the above five implicit models underpins the project.

A further important element of a *quality process* will be the identification and *allocation of roles and responsibilities* to appropriately skilled team members to enable the requirements of the methodology to be executed.

As an alternative to *process*, *measurement* is a classic approach to quality, whether that addresses process or product dimensions. For instance, the ISO9126 *standard* specifies a set of software quality attributes, including: functionality; reliability; efficiency; usability; maintainability and portability [5]. Of course a standard may not be a high one (for instance I sometimes provocatively make the point by proposing that the food quality standard for a nameless global burger chain, is that the food tastes at least as good as the wrapper), but in the latter case it does provide a global basis for measurement of consistency. A set of *metrics* typically accompany such standards, as a yardstick by which conformance to the standard can be demonstrated or a lack of conformance can be highlighted. For students, standards for coding and document formatting, or for recording meeting minutes may be relevant examples, where compliance with the quality standard can be objectively demonstrated.

The typical notion that students begin with when considering QA is the idea of *testing*. However testing, while part of the QA armoury, is more properly classified as a *quality control* activity (QC) rather than QA. Rather than acting in an overarching role to help ensure quality – it is inherently part of the production function, but as a control check added on at the end. Testing is not adequate if used as the sole mechanism to assure quality. The favourite mantra of a software project manager with whom I worked many years ago was "Quality is built in not bolted on!"

Having made that point, it is nonetheless true that a well framed and multi layered testing strategy (including unit tests, integration tests, usability tests, performance and stress tests, acceptance tests etc.) is a key element supporting a QA framework for systems related projects.

In contrast to the rather defensive and backstop position occupied by testing in supporting QA activities, more in-line activities of *quality review* have much to offer. For instance Robert Glass when asked for the three best software engineering practices came up with "*inspections, inspections, inspections*" [9], arguing that they "do a better job of error-removal than any competing technology". The notion of *review* of work in progress, whether by peers, experts, or by feedback from clients has much to commend it, and can be applied to a wide range of project types. Even simple practices such as mandating a peer review cycle for any documentary artefact produced by the team can play a significant role in improving the quality of the end product. Reviews in turn can be *periodic* and *formalised* through mechanisms such as "a walkthrough and a formal technical review" [3], "design and code inspections" [7] or other forms of audit. But they may also involve more *continuous processes* such as pair programming, or the shared workshop models of Joint Application Design (JAD) [4]. Likewise Test driven development (TDD) [13], in which design of tests leads development work, can also be thought of as a *continuous review* process, whereby quality is "built in" from the outset.

Further specific *practices* may be applied in support of QA activities. For instance ongoing practices of continuous integration, regular (e.g. daily) code builds, refactoring etc. [8] may be adopted. Then more control oriented practices may be useful such as change control, configuration and version management [2].

Finally at a meta-level there is the notion of *continual process improvement*, sometimes termed software process and practice improvement (SPPI), which "aims to build an infrastructure and culture that support effective methods, practices, and procedures and integrate into the ongoing way of doing business" [1]. These process improvement models are typically supported by blueprints which provide guidance to the meta-processes to be followed. In turn the practice dimensions of these process improvement models may be realised through "Recipes for software practices…that blueprints specify" [1].

Few student projects will reach the meta-level of *reflection* inherent in a process improvement layer, but such a sophisticated level of quality awareness remains a goal to aspire towards. For meta-level thinking may be realised in part, as students reflect upon the effectiveness of the processes and practices they have applied in their projects. Ideally they would adapt and refine them as they proceed. At a minimum we would expect students at the end of their projects to reflect upon the processes and practices they have applied during their projects and demonstrate awareness of how they could have done things differently and what those improvements might look like in future.

Thus we can see not only the elusive nature of quality, but the complexity of quality assurance and the variety of techniques, processes and practices outlined here that may serve to build quality into a project.

[1] Aaen, I. Software process improvement: Blueprints versus recipes, *Software, IEEE*, vol.20, no.5, pp. 86- 93, Sept.-Oct. 2003.

[2] Allan, G. A critique of using grounded theory as a research method. *Electronic Journal of Business Research Methods, 2* (1) 2003, 1-10.

[3] Aurum, A., Petersson, H. and Wohlin, C. State-of-the-Art: Software Inspections After 25 Years. *Software - Testing, Verification and Reliability, 12* (3) 2002, 133-154.

[4] Carmel, E., Whitaker, R. D., and George, J. F. 1993. PD and joint application design: a transatlantic comparison. *Commun. ACM* 36, 6 (Jun. 1993), 40-48.

[5] Carroll, C. The Cost of Poor Testing: A U.S. Government Study (Part 1). *EDPACS - The EDP Audit, Control, and Security Newsletter, 31* (1) 2003, 1-17.

[6] Corder, M., Horsburgh, M. and Melrose, M. Quality Monitoring, Innovation and Transformative Learning. *Journal of Further & Higher Learning, 23* (1) 1999.

[7] Fagan, M. Design and code inspections to reduce errors in program development. *IBM Systems Journal, 3* 1976, 182-211.

[8] Fowler, M. *Refactoring - Improving the Design of Existing Code*. Addison Wesley Longman, Boston, 1999.

[9] Glass, R. Inspections - Some Surprising Findings. *Communications of the ACM, 42* (4) 1999, 17-19.

[10] Kitchenham, B. and Pfleeger, S. Software quality: the elusive target *IEEE Software, 13* (1) 1996, 12-21.

[11] Kitchenham, B. Procedures for performing systematic reviews. Technical Report Keele University TR/SE-0401 and NICTA 0400011T.1, Software Engineering Group, Department of Computer Science, Keele University, and Empirical Software Engineering, National ICT Australia Ltd., July 2004.

[12] *Lifecycle Services White Paper*. (2005). Retrieved March 18, 2010, from Cisco Systems, Inc.: http://www.cisco.com/warp/public/437/services/lifecycle/LifecycleServicesWhitePaper.pdf

[13] Sanchez, J., Williams, L. and Maximilien, E. On the Sustained Use of a Test-Driven Development Practice at IBM *Agile 2007 Conference,* Washington, D.C 2007, 5-14.