Software Quality Assurance for Software Engineers

Dr. Deepti Suri

Assistant Professor
Department of Electrical Engineering and Computer Science
Milwaukee School of Engineering
Milwaukee, WI 53202
suri@msoe.edu

Abstract: Software Quality Assurance (SQA) is an important aspect of Software Engineering (SE) but there exists a wide variety of ways in which this topic is covered in an undergraduate curriculum at various institutions. At some schools, SQA is typically taught as a "topic" in a software engineering course, whereas in some other curricula, there are entire courses devoted to this area. There also exist broad spectrums of topics that are covered in these courses ranging from preliminary testing types & techniques to testing process, test metrics, inspections, configuration management etc. This paper questions as to what are some of the essential topics that should be a requirement for an undergraduate software engineering curriculum and the rationale behind it. Various strategies on how SQA can be integrated vertically as well as horizontally throughout a "sound" curriculum are also discussed. Milwaukee School of Engineering has one of the first ABET accredited SE programs in the United States. This paper also describes how SQA is being taught at MSOE and discusses the advantages and limitations of our approach.

1. Introduction

A Joint Task Force on Computing Curricula (CC2001) was established in 1998 by Institute for Electrical and Electronic Engineers (IEEE-CS) and the Association for Computing Machinery (ACM) to undertake a major review of curriculum guidelines for undergraduate programs in computing. The task force was assigned the task of providing curriculum guidance in the areas of (i) Computer Science, (ii) Computer Engineering, (iii) Software Engineering and (iv) Information Systems, so that the enhanced curriculum takes into account the latest developments in computing of the past decade and can be relevant through the next decade. The Steering Committee that was responsible for the area Software Engineering was called Computing Curricula Software Engineering (CCSE).

The CCSE Steering Committee has been actively working to address the software engineering curriculum issues at the undergraduate level. The CCSE Steering Committee is composed of representatives from the IEEE-CS and ACM as well as the Australian Computer Society, the British Computer Society, and the Information Processing Society of Japan. In addition, committee members are from several countries including Australia, Canada, Israel, Japan, the United Kingdom, and the United States.

One of the responsibilities of the CCSE Steering committee was to define and document a software engineering body of knowledge appropriate for guiding the development of undergraduate software engineering curricula. The body of knowledge is called Software Engineering Education Knowledge (SEEK), which published its first set of guidelines in fall 2002. Since then SEEK has been reviewed and revised and the final draft is available at http://sites.computer.org/ccse. It is important to note that SEEK does not represent/provide a curriculum but rather provides the foundation necessary to build a sound software engineering curriculum. We in this paper concentrate on "Software Quality Assurance" as it pertains to an undergraduate curriculum.

It has been recognized early on [1] [2] [5] [6] that to better prepare our students for the future, we have to introduce quality techniques and activities early on. Unfortunately, most academic activities concentrate on development of products (i.e. languages like Java, C++, OO concepts etc.) rather than a process that should be used to develop high quality products. At some places, where issues of quality assurances are tackled in the curriculum, the prime emphasis seems to be in testing, especially unit testing. Students end up (unfortunately as may professionals do too), approaching testing as an ad-hoc trial-and-error technique used at the end, after the development of the product.

SEEK has identified 10 areas which should be emphasized in the Software Engineering curriculum at the undergraduate level. Out of these, there are several (indicated by an asterisk *) that cover topics that traditionally fall under the umbrella of "Software Quality Assurance".

- Fundamentals
- Professional Practice
- Requirements *
- Design
- Software Construction *
- Software Verification and Validation *
- Software Evolution
- Software Process *
- Software Quality *
- Software Management *

2. Software Quality Assurance (SQA) in an Undergraduate Curriculum

At most educational institutions, courses are structured such that the technologies/activities devoted to the development of products are emphasized. Hence, software quality is treated as a secondary concern and the belief that quality should be addressed exclusively in testing, at the end of development is propagated. There is a widespread realization in both industry and academia that this belief has to change. Software Quality should not be an afterthought but should rather be addressed at the front-end of the life cycle. Software development curricula should focus on quality and the concepts of quality should be integrated well into the undergraduate curriculum.

The proactive institutions who have decided to incorporate quality concepts into the curriculum currently are struggling with the format/content that they should be addressing in these courses. Based on the SEEK objectives, the topics that the students should be taught/exposed to are

detailed in the Appendix. As can be seen, SEEK has managed to churn out a very relevant and comprehensive list of topics that should be covered in an undergraduate curriculum, but has not provided any template on how these topics should be covered.

The current challenge is that there exist a wide variety of ways in which these topics are covered in an undergraduate curriculum at various institutions and certainly not all of them can be covered in an undergraduate curriculum. At some schools, SQA is typically taught as a "topic" in a software engineering course, whereas in some other curricula, there are entire courses devoted to this area. There also exist broad spectrums of topics that are covered in these courses ranging from preliminary testing types & techniques to testing process, test metrics, inspections, configuration management etc. At least at the moment, there does not seem to be a consensus in the academic community on what are some of the "typical" skills that are expected of "typical" software engineer.

3. Software Quality Assurance at Milwaukee School of Engineering

The academic schedule at MSOE is based on a quarter system with three quarters in an academic year. Each quarter involves ten weeks of instruction with the eleventh week devoted to final exams. Typical software engineering courses are three or four credits, and most have an associated laboratory session. The undergraduate software engineering program at MSOE [4] began operation in 1999 and had its first graduating class in spring 2002. The SE program was visited by the Accreditation Board for Engineering and Technology (ABET) in September 2002 and is one of the first accredited SE programs in the United States.

The language of choice, in which most of the current development is being done is C++. The software engineering students take a two course programming sequence [Computer Programming (CS-182[1]) and Software Design (CS-183)] which introduces them to programming and object oriented concepts. At the end of the two course sequence, the students are familiar with the OO concepts of encapsulation, inheritance, polymorphism etc. In the version 2.1 of the curriculum that goes into effect in the academic year 2004-2005, the two course programming sequence will be replaced by a three course programming sequence in Java.

The students take two courses dedicated to the area of Quality Assurance as part of their curriculum (i) Introduction to Verification (SE-283) as sophomores and (ii) Software Quality Assurance (SE-4831) as seniors. These courses meet twice a week for fifty minutes each for lecture and for two hours once a week for a lab session.

In addition, a lot of quality assurance concepts are introduced and practiced in their three-quarter experience of "Software Development Laboratory" [3], where students work on large scale projects incorporating industrial practice in an academic setting.

SE-283 was a sophomore course that was run for the first time in the fall quarter of 2003. In this course, the focus is on introducing students to software testing and the integration of testing into the software development process. Topics covered include basic testing techniques, designing for testability and use of version control systems. Students are shown various strategies of unit testing and are also exposed to CPPUnit (a tool used for unit testing C++ classes). They were also exposed to the areas of requirements analysis and testing with special emphasis on how to

[1] The course numbers correspond to version 2.0 of the software engineering curriculum at MSOE.

develop a testing outline (not as formal as a test plan yet) and then develop test cases from it. Concepts of manual vs. automated testing and status reporting are also covered briefly.

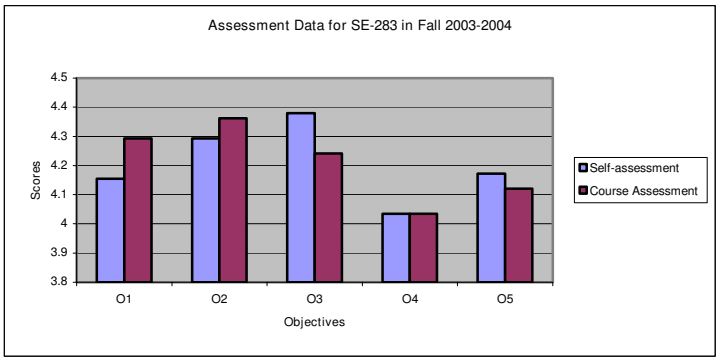
The concepts introduced in the lecture are reinforced using lab projects. In the first offering of he course, the students worked on three lab projects throughout the quarter. The first project focused on using CPPUnit to test various classes that students had written. This provided the students an opportunity to apply some of the unit testing strategies that were discussed in class. The second project focused on using client requirements to develop a test outline and test cases and then using those test cases to determine the quality of a project. Initially, the students were only provided the requirements and had to come up with a testing strategy. Subsequently, they were provided with the executable and had to execute their testing strategy on the executable and report on the quality of their project. As a last step, they were given access to the source code and were asked to fix some of the bugs that they had discovered earlier. The third project focused on using and the needs for a configuration management tool for a project. For the third project, the students worked in groups of three, where each one had a specific task but was dependent on the code produced by the others. The only way that the group members could communicate was through their CVS repository.

Our preliminary assessment reveals that the students in the class were divided into two camps: (i) People who thought that the material was interesting and relevant to Software Engineering and (ii) People who found the material very boring and useless. In general, I believe that the students (sophomores) did not have the maturity to understand the relevance and impact of testing on the quality of a good software product. The students who understood the impact found the class relevant. Nevertheless, we have to do a better job communicating that to them. At MSOE, each course has a set of objectives that are published and distributed to the students at the beginning of the term. At the end of the term, students are asked to evaluate themselves on how successful they were at meeting each objective and then evaluate the course on helping them meet those objectives. The rating used is 1 (not successful at all) to 5 (very successful). The course objectives for this course are presented in Table 1. The average scores of all the students for all the eight objectives are presented in Figure 1.

Table 1: Course objectives for SE-283

O1	Be able to integrate testing into the software development process.
O2	Understand the role of testing and configuration management in the verification and validation of software
O3	Be able to understand the importance of manual and automated testing.
O4	Be able to understand the importance of reliability testing and acceptance criteria.
O5	Be able to communicate test specifications, analysis, and results in written and oral form.

Figure 1: Assessment Data for SE-283



From Figure 1, we see that students believe that they were reasonably successful in meeting Objectives 1 and 2 (scores > 4) and the course played a significant role in helping them achieve these objectives. This agrees with our (as the instructor of the course) assessment of the course too. For Objectives 3 and 5, the students believe that they did very well. In-fact, they believe that they did better than what the class helped them achieve. This was not a surprising result considering that their lab projects reinforced the concepts we talk about in these objectives. It is a well known pedagogical observation that students learn better when they "do things themselves". Regarding Objective 4, we believe that that we talked more about acceptance criteria than reliability testing in our classes and hence the students were not sure if they met the objective and if the class helped them meet the objective either.

In the previous version of our curriculum (version 1.2), there was only one course dedicated to quality assurance. The course was called Software Verification and Validation (SE-483). This course was primarily dedicated to testing topics (testing strategies, regression and acceptance testing, test management etc.) but concepts like version control, configuration management, inspects, reviews, testing process, formal methods etc. were ignored. Our assessment (primarily through surveys results provided by graduating seniors) revealed that we were not doing an adequate job in this area and hence in the revised version of the curriculum (version 2.0), there are now two courses (as mentioned earlier) dedicated to this topic.

It is important to note that SE-4831 has not yet been run at MSOE and its topical coverage is currently under debate and consideration. Hence, assessment information on this course will not be available till the Winter Quarter of 2005.

Software Development Laboratory (SDL) is a three-quarter course sequence in the junior and senior years, designed to provide the students a "real world" experience in an academic setting. This arrangement provides students an opportunity to work in teams on ongoing large-scale projects [3]. This setting also provides an opportunity to students to apply the quality assurance techniques that they have learnt in various courses to a project at different phases of a project life cycle. In addition to development responsibilities, students work on various "staff teams" such as the Software Engineering Process Group (SEPG), the Software Configuration Management group (SCM), the Software Quality Assurance team (SQA), the Planning and Tracking group (PT) and the Training Department (TD). The staff groups dedicated primarily to quality assurance issues are obviously SCM and SQA.

The SCM team is responsible for ensuring that configuration management and version control practices are followed, including proper use of the CVS repositories. The have come up with version control policies, tagging and logging standards to be used in the lab and continue to work on improving these policies. The purpose of the SQA staff team is to ensure that all development and staff teams follow a quality process and develop quality work products. Quality is ensured through work product and process audits. SQA also advises SEPG on opportunities for process improvement based revealed by these audits.

SDL is the course where all the development and quality techniques/issues that the students have seen in different courses "come together" for them. The students have consistently rated their experience in the SDL as one of their "most valuable" experience at MSOE for this reason.

4. Summary

This paper summarizes the author's experience of how the issues of software quality are tackled at her institution. This paper is written with the purpose of starting a debate of how Software Quality should be and can be taught in the various academic institutions. It is also meant to start a debate as to what are some of the core technologies and skills that a software engineer should possess when they graduate with an undergraduate degree in Software Engineering.

5. References

[1] Hilburn, T.B. and Towhidnejad, M., "Software Quality: A Curriculum Postscript", *Proceedings of the Thirty-first SIGCSE Technical Symposium on Computer Science Education*, Austin, TX, USA, March 8-12, 2000, pp. 167-171.

[2] Parnas, D., "Inspection's role in Software Quality Assurance", *IEEE Software*, V20, July/August 2003, pp. 16 – 20.

[3] Sebern, M.J., "The Software Development Laboratory: Incorporating Industrial Practice in an Academic Environment", *Proceedings of the 15th Conference on Software Engineering Education and Training*, Covington, KY, USA, February 25-27, 2002, pp. 118 - 127.

[4] Sebern, M. J. and Lutz, M. J., "Developing Undergraduate Software Engineering Programs," *Proceedings of the 13th Conference on Software Engineering Education & Training*, Austin, TX, USA, March 2000, pp. 305-306.

[5] Towhidnejad, M., "Incorporating Software Quality Assurance in Computer Science Education: An experiment", *Proceedings of the 23rd Annual Frontiers in Education Conference*, Boston, MA, USA, November 6-9, 2002.

[6] Voas, J., "Assuring Software Quality Assurance", *IEEE Software*, V20, May/June 2003, pp. 48-49.

[7] Web page for SEEK, http://sites.computer.org/ccse/

DEEPTI SURI

Deepti Suri is Assistant Professor of Software Engineering in the Department of Electrical Engineering and Computer Science at the Milwaukee School of Engineering (MSOE). She primarily teaches Software Engineering courses in the areas of Software Requirements and Specifications, Object Oriented Design, Design Patterns and Verification and Validation.

Prior to joining MSOE, Dr. Suri worked in industry for seven years. She has provided systems solutions for the Electronic Design Automation (EDA), financial, and health-care industries. Her experience includes working in all aspects of the project's life cycle.

Ms. Suri holds three degrees B.S. (1989), M.S. (1991), and Ph.D. (1999) in Computer Science. She has written several articles in the areas of Robotics, Parallel computing and Software Engineering Education and presented her work at national as well as international conferences.

Appendix

Enclosed below are the various Software Quality Assurance topics that are in one or more of the SEEK areas. Note that the entire draft can be found at http://sites.computer.org/ccse/

Values in the 3rd column indicate the Bloom's attribute for the particular topic: (k)nowledge, (c)omprehension and (a)pplication. Values in the 4th column are an indication of the topic's relevance to the core material: (e)ssential, (d)esirable and (o)ptional.

SEEK AREA: REQUIREMENTS

REQ.mgt	Requirements management		
REQ.mgt.1	Change management	c	E
REQ.mgt.2	Tracing	c	E
REQ.mgt.3	Special management concerns (e.g. consistency management, release planning, reuse, etc.)	k	E

SEEK AREA: SOFTWARE CONSTRUCTION

CON.lan	Language-oriented issues		
CON.lan.4	Assertions, design by contract, defensive programming	a	E
CON.tec	Construction technologies		
CON.tec.5	Error handling, exception handling, fault tolerance, and security	a	E
CON.tec.6	State-based and table driven construction techniques	a	E
CON.tec.7	Run-time configuration and internationalization	a	E
CON.tec.13	Hot-spot analysis and performance tuning	k	E
CON.tec.15	Test-first programming		D
CON.tl	Software Construction Tools		
CON.tl.3	Unit testing tools	c	E
CON.tl.4	Profiling, performance analysis and slicing tools		D

SEEK AREA: SOFTWARE VERIFICATION AND VALIDATION

VAV.fnd	V&V terminology and foundations		
VAV.fnd.1	Objectives and constraints of V&V	k	E
VAV.fnd.2	Planning the V&V effort	k	E
VAV.fnd.3	Documenting V&V strategy, including tests and other artifacts	a	E
VAV.fnd.4	Metrics & Measurement (e.g. reliability, useability, performance, etc.)	k	E
VAV.fnd.5	V&V involvement at different points in the lifecycle	k	E
VAV.rev	Reviews		
VAV.rev.1	Desk checking	a	E
VAV.rev.2	Walkthroughs	a	E
VAV.rev.3	Inspections	a	E
VAV.tst	Testing		
VAV.tst.1	Unit testing	a	E
VAV.tst.2	Exception handling (writing test cases to trigger exception handling; designing good handling)	a	E
VAV.tst.3	Coverage analysis (e.g. statement, branch, basis path, multi--condition, dataflow, etc.)	a	E
VAV.tst.4	Black-box functional testing techniques	a	E

VAV.tst.5	Integration Testing	c	E
VAV.tst.6	Developing test cases based on use cases and/or customer stories	a	E
VAV.tst.7	Operational profile-based testing	k	E
VAV.tst.8	System and acceptance testing	a	E
VAV.tst.9	Testing across quality attributes (e.g. usability, security, compatibility, accessibility, etc.)	a	E
VAV.tst.10	Regression Testing	c	E
VAV.tst.11	Testing tools	a	E
VAV.tst.12	Deployment process		D
VAV.hct	Human computer user interface testing and evaluation		
VAV.hct.1	The variety of aspects of usefulness and usability	k	E
VAV.hct.2	Heuristic evaluation	a	E
VAV.hct.3	Cognitive walkthroughs	c	E
VAV.hct.4	User testing approaches (observation sessions etc.)	a	E
VAV.hct.5	Web usability; testing techniques for web sites	c	E
VAV.hct.6	Formal experiments to test hypotheses about specific HCI controls	D	
VAV.par	Problem analysis and reporting		
VAV.par.1	Analyzing failure reports	c	E
VAV.par.2	Debugging/fault isolation techniques	a	E
VAV.par.3	Defect analysis	k	E
VAV.par.4	Problem tracking	c	E

SEEK AREA: PROCESS

PRO.con	Process concepts		
PRO.con.5	Software engineering process improvement (individual, tem)	c	E
PRO.con.6	Quality analysis and control (e.g. defect prevention, review processes, quality metrics, root cause analysis, etc.)	c	E
PRO.imp	Process Implementation		
PRO.imp.3	Life cycle process models and standards (e.g., IEEE, ISO, etc.)	c	E
PRO.imp.4	Individual software process (model, definition, measurement, analysis, improvement)	a	E
PRO.imp.5	Team software process (model, definition, organization, measurement, analysis, improvement)	a	E
PRO.imp.6	Process tailoring	k	E
PRO.imp.7	ISO/IEEE Standard 12207: requirements of processes	k	E

SEEK AREA: QUALITY

QUA.cc	Software quality concepts and culture		
QUA.cc.1	Definitions of quality	k	E
QUA.cc.2	Society's concern for quality	k	E
QUA.cc.3	The costs and impacts of bad quality	k	E
QUA.cc.4	A cost of quality model	c	E
QUA.cc.5	Quality attributes for software	k	E
QUA.cc.6	The dimensions of quality engineering	k	E
QUA.cc.7	Roles of people, processes, methods, tools, and technology	k	E
QUA.std	Software quality standards		
QUA.std.1	The ISO 9000 series	k	E
QUA.std.2	ISO/IEEE Standard 12207: the "umbrella" standard	k	E

QUA.std.3	Organizational implementation of standards	k	E
QUA.std.4	IEEE software quality-related standards		D
QUA.pro	Software quality processes		
QUA.pro.1	Software quality models and metrics	c	E
QUA.pro.2	Quality-related aspects of software process models	k	E
QUA.pro.3	Introduction/overview of ISO 15504 and the SEI CMMs	k	E
QUA.pro.4	Quality-related process areas of ISO 15504	k	E
QUA.pro.5	Quality-related process areas of the SW-CMM and the CMMIs	k	E
QUA.pro.6	The Baldridge Award criteria for software engineering		O
QUA.pro.7	Quality aspects of other process models		O
QUA.pca	Process assurance		
QUA.pca.1	The nature of process assurance	k	E
QUA.pca.2	Quality planning	a	E
QUA.pca.3	Organizing and reporting for process assurance	a	E
QUA.pda.4	Techniques of process assurance	c	E
QUA.pda	Product assurance		
QUA.pda.1	The nature of product assurance	k	E
QUA.pda.2	Distinctions between assurance and V&V	k	E
QUA.pda.3	Quality product models	k	E
QUA.pda.4	Root cause analysis and defect prevention	c	E
QUA.pda.5	Quality product metrics and measurement	c	E
QUA.pda.6	Assessment of product quality attributes (e.g. useability, reliability, availability, etc.)	c	E

SEEK AREA: MANAGEMENT

MGT.cm	Software configuration management		
MGT.cm.1	Revision control	a	E
MGT.cm.2	Release management	c	E
MGT.cm.3	Tool support	c	E
MGT.cm.4	Builds	c	E
MGT.cm.5	Software configuration management processes	k	E
MGT.cm.6	Maintenance issues	k	E
MGT.cm.7	Distribution and backup		D