SOFTWARE QUALITY ACROSS THE CURRICULUM

Thomas B. Hilburn[1], and Massood Towhidnejad[2]

***Abstract* —. *One of the major areas of software engineering, as specified in the SoftWare Engineering Body Of Knowledge, is software quality. This paper argues that software quality should be a central focus in an undergraduate computer science or software engineering curriculum, and that quality should not be viewed only as an exercise in software testing. First, we discuss the concepts and techniques that are needed to develop a high-quality software product. Then, we propose a model for viewing software quality and use the model to advocate quality activities that should be incorporated in a curriculum, addressing both product quality and process quality. We also share some of our ideas and experiences about integrating quality activities into the curriculum. Finally, we describe why an undergraduate software engineering curriculum should include at least one dedicated course in software quality. We present a detailed outline of such a course.*

Index Terms — Software Engineering, Software Process, Software Quality, Undergraduate Curriculum.

INTRODUCTION

Look about you: the electrical shaver that you used this morning, the TV that gave you the latest news, the car that you drove to work, and the microwave that heated your lunch - all have software that is responsible for control of these operations. By the way, have you taken a flight lately? - There are almost four thousand processors and two million lines of code controlling a Boeing 777 aircraft. This does not even take into account software that supports the plane's flight: code for the air traffic control system and the navigation satellites. Clearly, our daily activities strongly depend on software, and soon we will have personalized robots that will be part of our households and will supply even more support and control of our day-to-day life. If we do not pay attention to the issue of software quality, this could be a very scary picture.

Now take a look at the software industry - with the exception of safety critical software, where people lives are at stake, industrial software does not have high quality. (e.g., When was the last time you saw a blue screen on you computer?) However, we should make it clear, when we talk about software quality, we are not just interested in whether software fails, but we also interested in measurement and assessment of software quality, and the method, techniques, and the processes needed to produce a quality product in an effective and efficient manner. And, of course we want to discuss how this affects curriculum design and pedagogy. We will expand on these issues in the later sections.

Fortunately, industry has started responding to the customer's outcry for better quality software. Latest industry data indicates they are spending over fifty percent of their project budget on activities that would increase the quality of their software. Unfortunately, most companies spend these resources in the testing phase, chasing defects that have already been introduced in the front-end part of the development life cycle. This is a waste of time and resources, and industry leaders have shown that they can reduce testing and warranty costs by more than half, when they start implementing practices that forces software quality control throughout the software development life cycle, with special emphasis in the earlier stages of development. For example, a study produced by the Jet Propulsion Laboratory (JPL) [7] shows us that the cost of defect detection and correction increases ten fold as we move through each phase of the life-cycle (requirements to design to implementation/test to release). Hence, the JPL study indicates the ratio of defect costs is 1:10:100:1000 (requirement : design : implementation/test : post release). This means a defect injected during the requirements phase would cost 1000 times more to fix during the post release than in the requirement phase.

We, as educators, need to be honest and share part of the blame for the state of the software industry, and the lack of high quality software. Computer science curricula do not typically address software quality issues in a meaningful way, and do not provide educational units on how to build high-quality software products or how to integrate quality activities throughout the software development process. Rarely will you see a, computer science curriculum offer a dedicated course on software quality, and throughout the curriculum, whenever software quality is addressed it is under the umbrella of testing or debugging code. This is very obvious from the latest literature (see recent issues of *SIGCSE Bulletin*, *Computer Science Education*, and the *Proceedings of the Conference on Software Engineering Education and Training*). Therefore, we conclude that students graduating with a bachelor's degree in computer science lack the necessary knowledge and experience in the area of software quality. However, the first job assignment of about thirty percent of the students graduating with a degree in computer science is in an area that is closely related to software quality (i.e. software testing, software maintenance, etc.) Employers of these graduates often

[1] Thomas B. Hilburn, Embry-Riddle University, Department of Computing, Daytona Beach, FL 32114, hilburn@erau.edu
[2] Massood Towhidnejad, Embry-Riddle University, Department of Computing, Daytona Beach, FL 32114, towhid@erau.edu

complain about a new hire's lack of knowledge in the area of software quality.

This situation closely resembles the situation we as educators faced about ten years ago. At that time, the software industry was asking us to incorporate more software engineering into our curricula. Ten years later, we see that more and more computing programs are incorporating software engineering as part of their required courses, and more recently we see the establishment of new undergraduate degrees in software engineering. With the recognition by ABET that software engineering is a recognized curriculum area, and the forthcoming guidance from Computing Curriculum 2001 for software engineering education, it is obvious that the software engineering degree is here to stay. We as educators have the responsibility for delivering computing and software engineering curricula that ensure that our students are capable of developing high-quality software products. In the remainder of this paper, we discuss our view of software quality, and how it can be introduced in the undergraduate curriculum, either as a dedicated course or as an integral part of other courses.

CONCEPTS AND PRACTICES

Software Quality is something that everyone is in favor of and supports, at least in principle. However, when we get down to the questions of "What is software quality?", "How do we obtain it?", "How do we know when we have it?", and "What does it cost?", there is often confusion and a lack of agreement, both in academia and industry, on how to answer these questions. Actually, there should be little confusion about these issues. Although software quality is not a simple issue, its concepts and practices have been thoroughly studied and documented. For example, the *SWEBOK* (SoftWare Engineering Body OF Knowledge) [3] has two chapters directly addressing software quality knowledge, Software Quality and Software Testing, which provide a detailed overview of quality concepts and practices, along with an extensive list of pertinent references. In our discussion of quality issues we will focus not only on the quality of the product, but also the quality of the process used to develop the product.

The first step in determining the meaning of quality is to determine the necessary and desired quality attributes of the software product to be developed. Generally, product quality attributes can be classified as functional, reliability, usability, efficiency, maintainability, or portability. Although all of these might be considered in the development of a particular product, certainly "functionality" would be considered in all cases, although there is often an emphasis on functionality with little attention to other quality characteristics.

In order to achieve the quality goals for a product, a process must be set in place that defines, organizes and assesses the required quality assurance activities. Quality activities cover a whole spectrum of verification and validation (V&V) techniques, which include inspections, reviews, tests, and audits of software artifacts produced in the development process. Although industry is paying increasing attention to quality activities and processes, this area has largely been ignored in academia.

Finally, to determine whether the process is effective in reaching the quality goals, it is necessary to define quality measures, measure and collect the data, and use the data to analyze and assess the achievement of the quality goals. This data can also be used to assess the processes used and the costs of the quality activities as related to the overall development costs.

CURRICULUM ISSUES

In the last few years there has been a flurry of interest in updating exiting computing curricula and creating new programs in software engineering education. A task force of the ACM and IEEE Computer Society has recently produced a set of curriculum materials [1] for computer science curricula and there is another ACM/IEEE-CS task force at work on a similar volume dedicated to software engineering programs. In 1999, the Working Group in Software Engineering Education and Training produced the *Guidelines for Software Engineering Education* [2], which provides guidance for development of undergraduate programs that emphasize software engineering.

Although these efforts are bringing attention to the issue of software engineering education and they do provide overall structure and support for the design and implementation of software engineering oriented curricula, we still do not believe there is sufficient attention to what we believe should be a focus in software engineering education: software quality. We propose a model for software engineering education, that represents a modification of the V-Model for software development [8], which emphasizes the integration of software quality issues throughout the software development life-cycle.

A Quality Model

Figure 1 depicts the principal elements of the so-called "V-Model for Quality." The rectangular elements in the figure represent the traditional phases in software development (requirements and design on the left and testing on the right). This model contains features that have significant implications and ideas for designing curricula, which will help students learn how to produce high-quality software products in an effective manner. First, and most obvious is that the model advocates that one carry out a problem analysis and high-level design before proceeding to implementation and testing. Unfortunately, there are so many examples, both in industry and academia, of the emphasis on a code-and-test mentality, that we need to state the obvious: concentration on the front-end part of development can produce a higher-quality product at a lower cost. [5]

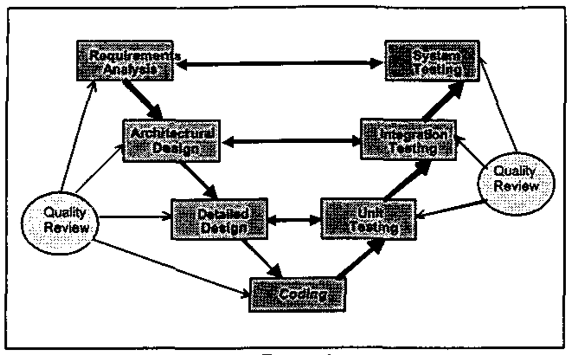

FIGURE 1
V-MODEL FOR QUALITY

Next, notice the double-headed arrows connecting the development phases on the left and right. The arrow pointing to the left signifies dynamic testing of the software against a software artifact (system testing of the software system against the requirements specification, integration testing using the architectural design, and testing of a unit against the unit design). The arrow pointing to the right represents the test planning that should take place in parallel with a development phase (e.g., system test planning should take place in conjunction with the development of a requirements specification). The curriculum implications of this part of the model are as follows: students should learn about and do system test planning at the same time as they learn how to analyze and specify software requirements; teaching integration planning should correspond with the teaching of high-level design; and unit test planning should be taught along with detailed design of a unit.

Finally, the "quality review" bubble in the model emphasize that the deliverable(s) produced in each development activity (software requirements specification, software design specification, code, system test plan, integration plan, unit test plan) should be subject to a quality review.

Processes for Software Quality

Although the V-Model for Quality provides an outline of what a software quality process should contain, it does not provide the detail and support needed to implement such a model in an academic setting. There do, however, exist processes that are useful in this area.

To address industrial and academic problems in software development, the Software Engineering Institute, Carnegie Mellon University (SEI) has developed a family of process improvement methods for individuals, teams, and organizations. The Personal Software Process (PSP)[3] was developed to help students and professional software engineers to organize and plan their work, track their progress, manage software quality, and analyze and improve their personal performance [5]. The PSP incorporates the "lower" part of the V-Model for Quality: it includes test planning as one of the design phase activities, and it includes two formal reviews, one for design and one for code (using a defined review process, development standards, and review checklists). Introductory and graduate PSP courses are now being used by over thirty institutions. The PSP provides an introduction to the concepts and techniques for producing quality software; and it sets the foundation for more advanced processes and practices.

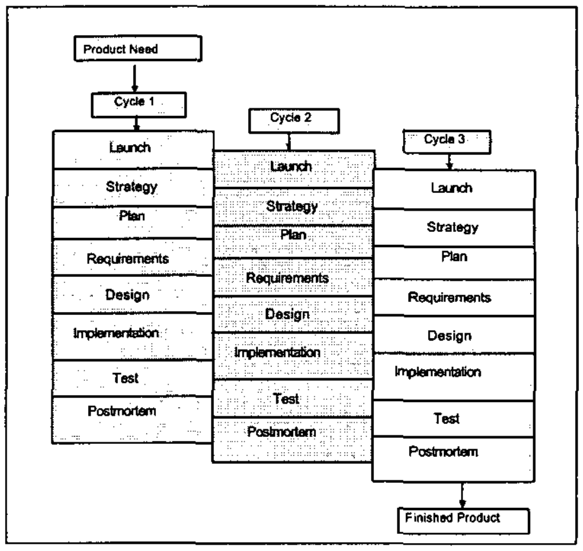

FIGURE 2
TSPi STRUCTURE AND FLOW

The SEI has also developed the Team Software Process (TSP)[4], a defined software process, based on the PSP, which guides and supports a team of software engineers in developing quality software in an effective manner. It provides guidelines, process scripts, tools, methods and techniques for developing a software product by a team. It is based on a sequential incremental model that divides development effort into a set of "development cycles" where each cycle involves producing software that satisfies some subset of the software requirements. The final cycle encompasses integration and test of the entire system (all requirements satisfied). In 2000, the SEI developed the TSPi, an academic version of the TSP [6], which is now used in over twenty computing degree programs. Figure 2 depicts TSPi cyclic model that forms the framework for the process.

The TSPi can be viewed as an embodiment of the quality concepts and practices we discussed in the V-Model

[3] Personal Software Process and the PSP are service marks of Carnegie Mellon University.

[4] Team Software Process and TSP are service marks of Carnegie Mellon University.

for Quality. It advocates test planning in conjunction with the requirements and coding phases, and it incorporates inspections, reviews, and audits in each phase, in each development cycle. In addition, both the PSP and TSPi emphasize quality planning, which is supported by a set of metrics and benchmarks that can be used for goal setting and analysis of results. Table 1 lists some of these metrics.

TABLE 1
PSP/TSPI METRICS

Metric Type		Description
Product Quality	Total defects/KLOC	the number of defects found in development, per 1000 lines of code
	Test defects/KLOC	the number of defects found in test, per 1000 lines of code
Process Quality	Yield	the percent of defects removed prior to the first compile
	Appraisal COQ (Cost of Quality)	the percent of total development time spent in design review and code review
	Failure COQ	the percent of total development time spent in compile and test
	Total COQ	Appraisal COQ + Failure COQ
	A/F R	$\frac{\text{Appraisal COQ}}{\text{Failure COQ}}$
	Review rate - LOC/hour	the number of lines of code reviewed per hour in a review (code review or design review)
	Defect removal rate - defects/hour	the rate at which defects are removed in a defect removal phase (requirements inspection, design review, code review, compile, test)

An Integration Strategy

We not only believe that software quality should be a central element of computing curricula, but we believe it can be done at a rather low cost (without adding lots of additional hours to the curriculum or significantly changing current curriculum content). This does not mean that it is easy or that it does not take a great amount of effort to incorporate quality features into an educational program. What follows is a brief outline of how this may be done and what sort of resources it will take.

- First and most important, the program faculty must agree that a focus on software quality is important. We suggest the faculty educate themselves (or get some training) in the PSP and the TSPi. This has two benefits: the processes illustrate the ideas presented in this paper and can help motivate interest in curriculum change, and it prepares faculty to teach and use these processes in their courses.
- Introduce quality concepts and practices in the first course. At Embry-Riddle we have found that first year students can do early test planning, carry out code reviews, and measure and analyze the quality of their programs. We have incorporated these in the closed labs for CS1 and CS2 [4].
- Get students involved in team projects in the second or third year of their program – do not wait until the last semester of their senior year. Use a team software process and require your students to follow it. We like the TSPi; but whatever you use, it should include setting quality goals, formal reviews and inspections of software artifacts, early test planning, and measurement and analysis of the quality of the product and the process.
- Integrate software quality concepts and practices into every course that deals with some aspect of software development. Of course, in some courses, such as a senior software project course, it is easy to identify software quality topics and activities that can be included, but in others, such as a course in operating systems, it is less obvious. Here is an example of something you could do in an operating systems course: you might spend part of a lecture hour discussing the cost of making a change to an existing operating system (How long will it take to make the change? Is the historical data available to make estimates about such costs? If an error is introduced in the OS and it is distributed to the public, what is the impact? How would one make sure there were no errors introduced during the maintenance work?). Just getting students to ask the right questions about quality can be a significant contributor to the understanding of software quality issues, and at little added cost to the curriculum.

A COURSE IN SOFTWARE QUALITY

As was discussed in the previous section, one could integrate quality activities throughout the undergraduate curriculum; this approach reinforces the issue of quality over a number of courses. This means that, as students progress through their program of study, they will be introduced to a variety of quality activities. The advantage of this arrangement is that the appropriate quality activity is introduced in that course in which the quality activity is most relevant. The disadvantage of this approach is that almost all the courses in the curriculum have already been overloaded with topics that "have to be covered", and if we are not careful, we might not present the quality activities/topics in an appropriate level of detail and emphasis, which will result in the current situation, where quality topics will become an afterthought.

The second approach involves adding a course devoted to software quality. We recommend that the course be introduced in either the second semester of the junior year or during the senior year. The main purpose of this course is to highlight the importance of software quality assurance, and provide students with in-depth knowledge and experience about the concepts, techniques and processes associated with software quality. The pre-requisite knowledge for students entering this course should be an understanding and appreciation of the software development life cycle with an emphasis on software requirement specification and software design specification. In addition, students should have a good understanding of algorithms and data structures, and good programming skill in at least one high level programming language.

The breath and depth of the material covered in this course depends on the following factors: 1) what software quality topics have already been covered in the previous courses, and 2) what is the purpose of this course (e.g., provide a survey-like coverage of the breadth of quality topics, or to provide concentrated, in-depth coverage of one or two areas). Of course, the answer to these questions depends on the goals of the curriculum, and the background and interests of the faculty delivering the curriculum. The following sections describes a set of topics that could be covered in such a course; we recommend the faculty use these topics as a starting point, and design their course based on their curriculum goals and department resources. We would also like to discourage using this course as the only exposure that students have to software quality concepts and practices. We recommend some of these topics to be introduced in other courses, across the curriculum, and use this course to provide more in-depth coverage of software quality topics.

Topical Outline

In this section, we describe the major topics that could be covered in an undergraduate software quality course.

- Why Quality

This section covers the importance of quality. It also discusses cost and schedule problems caused by a lack of attention to software quality. In addition, this section draws a mental picture of what we mean by quality; specifically, we would discuss the common characteristics of quality, usually referred to as the "ilities" of quality (e.g., reliability, maintainability, etc.) Finally, a discussion of ethical issues that are related to quality should be covered under this topic.

- Quality of Product

In this section, various techniques and standards that are associated with product quality are studied. We divide the topics into three parts: 1) defect prevention, 2) testing, and 3) defect tracking.

Defect prevention covers techniques and activities that are designed to either prevent the injection of defects into an artifact, or capturing a defect before he final artifact is generated. Topics covered under defect prevention include: standards for different artifacts (i.e., requirements, design, and code), processes for generation of each artifacts (e.g., a requirements management process), and review checklists. Techniques covered under defect capture include; audits, reviews, walkthroughs, and inspections.

The second part covers the techniques and activities that are associated with testing. Topics that are covered in this area are: testing levels and techniques (i.e., unit, integration, system, regression, acceptance, installation, etc.), testing methods (i.e., black box, white box), and testing artifacts (i.e., test classes, test cases, test plan, etc.). Addition topics such as test coverage, trade offs, testing tools, test automation, and test organization are also discussed under this sub topic.

The defect tracking sub topic covers issues such as defect reporting, defect logs, and defect life cycle (where it was injected, where it was detected, when it was fixed, and what are its side effects).

We might note that an alternative approach to the study of software quality techniques would be to use a "static-dynamic" categorization. Dynamic techniques involve the execution of code, while static techniques do not. Another discriminating factor between these two categories is that static techniques are utilized in the early stages of development cycle, while dynamic techniques are incorporated after some portion of the code is completed. For example, reviews, walkthroughs, and inspections are static techniques, while traditional testing techniques fall under the dynamic category. As mentioned previously, this approach is implied by the V-Model of Quality.

- Quality of Process

Under the quality of the process, students should be introduced to different development processes, and gain a working knowledge of at least one such process. They should gain understanding of the relevant IEEE and ISO standards for software quality. In addition, topics such as planning, schedule and cost estimation, and configuration management would be studied as they relate to quality issues and concerns. Finally, a description of a Software Engineering Process Group (SEPG) and software process improvement methods are appropriate topics to be covered under process quality.

- Measurement

Measurement plays a key role in the study of software quality. The topics covered under measurement should include setting quality goals and how we can measure these goals. This area should cover the quality attributes to be measured, measurement techniques used to collect data, and the metrics derived form the measured data. The topics should also include data analysis methods and techniques. In addition, a brief introduction to causal analysis techniques should be included.

- Management issues

This section covers organizational issues associated with quality. Topics covered in this section includes how a quality organization could be set up, what are the roles and responsibilities of a quality assurance organization, the use of an external V&V group, and how an SEPG can help increase quality.

This course should include a series of hands-on experiments to re-enforce the topics covered. These experiments could fall into either individual or team activities. For example, students could develop test cases and test plans, execute these test plans, collect the test data, analyze the data, draw conclusions about software quality, and decide on appropriate action. In addition, students should participate in inspections and walkthroughs. We strongly recommend the artifacts produced by other students, in other courses, be used in these experiments (e.g., use a design specification developed in a project course).

CONCLUSION

In conclusion, we believe that software quality should be a major concern of software educators. We also think that the most effective way to accomplish this goal it to provide students with appropriate, defined software processes that support and guide the development of high-quality software. We suggest starting with a simple process and adding complexity as students mature and gain technical and analytical skills.

Software quality should not be an afterthought – software engineers need to address quality in the front-end of the software life-cycle; and in a like manner, educators need to introduce quality concepts and practices in the front-end of an academic program. This must then be followed with more in-depth learning about software quality concepts and practices. Only with this sort of attention and effort can we adequately prepare our students to build the high-quality software systems needed in today's society.

REFERENCES

[1] ACM/ IEEE-CS CC2001 project, *Computing Curricula 2001, Computer Science Volume,* December 2001.

[2] Bagert, D., et. al., *Guidelines for Software Engineering Education,* Version 1.0, Technical Report CMU/SEI-99-TR-032, Software Engineering Institute, Carnegie Mellon University, November 1999.

[3] Bourque, P. and Dupuis, R., eds., *Guide to the Software Engineering Body of Knowledge,* Trial Version 0.95, http://www.swebok.org/, May, 2001.

[4] Hilburn, T. and Towhidnejad, M., "Integrating Personal Software Process (PSP) Across the Undergraduate Curriculum", *Proceedings of 1997 Frontiers in Education Conference,* November 1997.

[5] Humphrey, W. S., *Introduction to the Personal Software Process,* Addison-Wesley, 1997

[6] Humphrey, W. S., *Introduction to the Team Software Process,* Addison-Wesley, 2000.

[7] Rothman, Johanna, "What Does It cost to Fix a Defect?", Column Archive, StickyMinds.Com, http://www.stickyminds.com/.

[8] *V-Model 97, Lifecycle Process Model* -Developing *Standard for IT Systems of the Federal Republic of Germany,* General Directive No., 250, http://www.scope.gmd.de/vmodel/en/, June 1997.