Thoughts on Teaching Software Quality Engineering

Witold Suryn
Department of Electrical Engineering
École de Technologie Supérieure Montréal, Canada
wsuryn@ele.etsmtl.ca

ABSTRACT

The article presents an overview of the subject of Software Quality Engineering (SQE) education. Four different perspectives are taken into account: why to teach SQE, how the subject is being taught today, what support teachers have to teach SQE and how could the Software Quality engineer be educated. The latest trends in methods and tools pertinent to the domain are also presented.

1. Introduction

Quality becomes a critical attribute of a software product since its absence results in dissatisfied users, loss of money, and – in the most critical situation – in lost lives. Increasing recognition of the importance of software quality makes the software engineering "center of gravity" shift from creating a technology solution toward a quality solution. Development organizations confronted with such an approach are, in general, not prepared to deal with it since their engineers too often are not adequately educated. And if such knowledge exists, it is still more of an empirical nature than resulting from a formal educational process. This reality makes the role played by Software Quality Engineering (SQE) education to change – or sometimes "about to change" - from "Cinderella" to a full member of Software Engineering Education family. This paper presents a quick look at today's situation in Software Quality Engineering education from four different points of view: why should SQE be taught, how is SQE being taught, what has an SQE teacher for his disposition and what and how could the future SQE engineer be educated. This paper has no ambition to cover the subject to its fullest extent, seeking rather a wide reaction from the community of academic teachers and researchers.

2. Why should Software Quality Engineering be taught

Applying a simplistic mathematic formula the answer could be presented in the following manner:

$$\textbf{RISK= F (1/quality)}$$

or,

The increased quality mitigates the risk

The ways in which the risk resulting from missing quality can manifest itself are numerous, however for a user the mostly recognized or experienced categories could be:

- Discomfort (ex: famous blue screens)
- Social or professional losses (information/data, time, money, job etc.)
- Loss of health or life (ex: over-X-rayed patients in one of Bristol hospitals)

It is rather obvious that our (i.e. users') dream about a bug-free, highest level quality solution for a small price will remain a dream for next *n* years, but how big this *n* will be, depends on the level of SQE education and knowledge that will find its way to software development companies. Most probably there is a chance to buy one day a software Rolls Royce but let's not forget that for car manufacturers it took a hundred years to obtain a sustainable high quality. Do we have to wait that long?

Missing quality takes its toll also from a developer/supplier even if his position is still so privileged that better word would be "untouchable". Until recently the biggest risk resulting from low or missing quality of software that a supplier could face was loosing a customer, money or market. An individual user was entirely helpless against a supplier whose low quality software created problems. But this situation evolves in the direction where a user or a group of users can seek compensation of their losses through legal actions [1]. Such an option adds two other risks to the list of supplier's risks:

- A court trial with a full range of consequences, and, when worse comes to the worst,
- An imprisonment.

It is then understandable why SQE receives a continuously increasing attention from software development companies. One of very positive consequences of the recent software-related legal evolution is an observable intensification of industry-funded joint research projects aimed to develop science and best practices in SQE [2, 3].

To conclude one could propose the following statement: *Software Quality Engineering should be taught because the time when both customers and suppliers will be fighting for best quality is knocking to our door*.

3. How is Software Quality Engineering being taught

How in reality SQE is being taught *today* only a teacher can answer so as many theories and practices as many teachers. Such a "liberty" is usually symptomatic to domains of knowledge that would obtain maximally CMM level "one-and-a-half" on the maturity scale. And that is true; SQE is immature, but not entirely. There are some considerable efforts aiming to create a structured approach to the SQE body of knowledge and teaching practices. First serious, professional efforts can be dated to early '90s [4] however in this paper only two most recent projects will be discussed.

Project 1: "Guidelines for Software Engineering Education" Version 1.0. Software Engineering Institute (USA), 1999. [5]

The document proposes the following structure of the body of knowledge of Software Engineering education:

- Core Area:
  - Software Requirements
  - Software Design
  - Software Construction
  - Software Project Management
  - Software Evolution
- Foundations Area
  - Computing Fundamentals
  - Human Factors
  - Application Domains
- Recurring Area (*Components in the Recurring Area are threads that occur through all of the Core Area components*):
  - Ethics and Professionalism
  - Software Processes
  - ***Software Quality***
  - Software Modeling

    - Software Metrics
    - Tools and Environments
    - Documentation
- Supporting Area that includes, but is not limited to general education, mathematics, […] and engineering

The proposed structure is well organized and offers a good support for those responsible for devising education curricula, however from SQE perspective of year 2003 this solution is hardly acceptable. Putting Software Quality in Recurring Area (refer to the explanation above) allows "gluing" this component freely wherever one could please. Moreover, it is practically infeasible to "glue" such a large component in one piece to another component, so in order to allow "gluing" the component has to be partitioned. How and following what logic?

Project 2: Software Engineering Education Knowledge (SEEK), IEEE Computer Society, ACM.

After SEEK, First Draft [6]:

"The education knowledge area group is responsible for defining and documenting a software engineering education body of knowledge appropriate for guiding the development of undergraduate software engineering curricula. This body of knowledge is called Software Engineering Education Knowledge or SEEK. The pedagogy focus group is responsible for using SEEK to formulate guidance for pedagogy as well as course and curriculum design to support undergraduate software engineering degree programs. The initial selection of the SEEK areas was based on the Software Engineering Body of Knowledge (SWEBOK) knowledge areas"

This innovative project with ambitions to become an international reference source for Software Engineering education programs as the first recognizes Software Quality as an individual knowledge area (KA) that requires individual teaching thread.

Phase: SEEK, First Draft, August 2002 [6].

Proposing Software Quality as the separate KA makes SEEK the most modern path leading to SQE maturity, but in spite of that the project is not free from critic. Quoting from [8]:

- 17 hours of core contact (hours of teaching) dedicated to all the content of Software Quality component in comparison with 250 hours for Software Engineering Fundamentals seems to be seriously underscored

- The presence of Software Quality Standards indicates that standards have finally become part of modern software engineering education; however the documents proposed as seminal do not always represent the state-of-the-art.
- There are subjects that did not yet find their place in SEEK's Software Quality KA, as f.e. Software Quality Management and Evaluation or Software Quality Implementation.

For more details on critic of SEEK, First Daft refer to [8].

Phase: SEEK Second Draft, December 2002 [7]

The critical opinions about First Draft expressed by researchers and teachers participating in the international conference on Software Technology and Engineering Practices (STEP) in Montreal, Canada in 2002 have been taken into account during redaction of Second Draft of SEEK project. The outcome however still leaves something to desire as the improvements seem to be rather cosmetic (ex: the number of teaching hours was increased from 17 to 21). For more details refer to [7].

In summary – SQE education is still being led by "heroes" (CMM level one) but what makes it reach the level "one-and-a-half" is SEEK. The only thing that has to be done now is to make SEEK also mature.

4. What can a teacher use to teach Software Quality Engineering

In year 2003 even scientific "heroes" do not stay without at least some support. In case of SQE this support is present on all crucial levels: theory, practice, tools and experience, but unfortunately is not very reach.

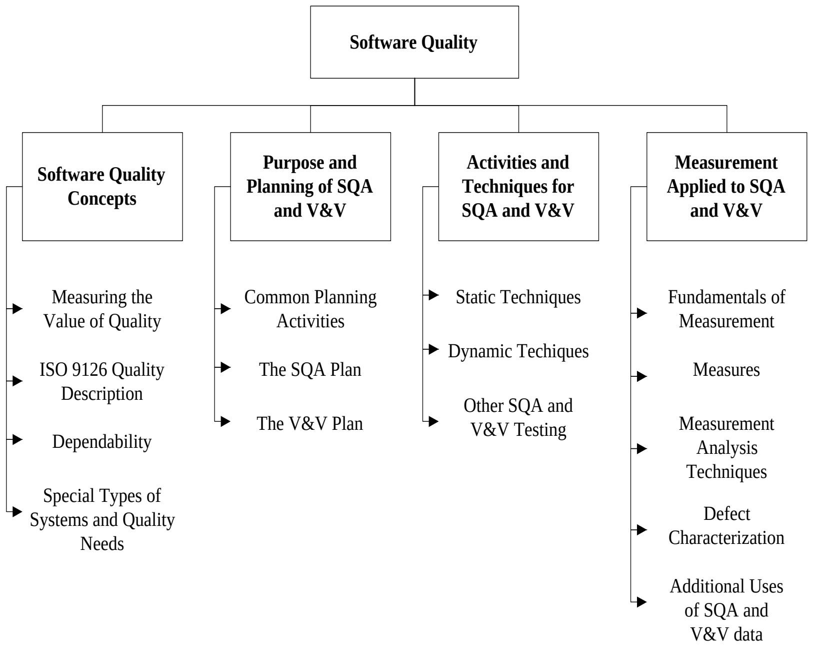
On the level of *theory* two major projects offer some support to an SQE teacher: SEEK and SWEBOK.

SEEK, as it was presented in the previous chapter, is an ambitious project with rather good provisions for success but still relatively far from completeness and stability.

SWEBOK (Software Engineering Body of Knowledge) [9] however is the project with over 5-year history, realized with participation of over 500 specialists worldwide aiming to create internationally recognized guide to Software Engineering Body of Knowledge. The results of this project were found so satisfactory that several educational institutions took SWEBOK as the basis for modernizing undergraduate Software Engineering programs [10, 11].

For SQE education SWEBOK offers:

- Recognition of Software Quality as a separate Knowledge Area
- Recommended Software Quality KA contents (several KAs are being under thorough revision what already resulted in proposed enhancements and improvements [12, 13]).
- Recommended breakdown of topics (Figure 1)
- Seminal references, including standards
- Further reading



Software Quality
Software Quality Concepts
Purpose and Planning of SQA and V&V
Activities and Techniques for SQA and V&V
Measurement Applied to SQA and V&V
Measuring the Value of Quality
ISO 9126 Quality Description
Dependability
Special Types of Systems and Quality Needs
Common Planning Activities
The SQA Plan
The V&V Plan
Static Techniques
Dynamic Techiques
Other SQA and V&V Testing
Fundamentals of Measurement
Measures
Measurement Analysis Techniques
Defect Characterization
Additional Uses of SQA and V&V data

Figure 1. SWEBOK's Software Quality Knowledge Area breakdown of topics

On the level of *practice* an SQE teacher is supported by several elements that represent so sought maturity and stability. Among these important elements are:

- Quality models: McCall, Boehm, Dromey [14], ISO/IEC 9126 [15]
- Software Product Quality Measurement and Evaluation standards
    - ISO/IEC 9126 [16, 17,18]
    - ISO/IEC 25000 SQuaRE (next generation of ISO 9126) [19]

- Software Product Quality support processes standards
    - ISO/IEC 14598 [20]
    - ISO/IEC 25000 SQuaRE (next generation of ISO/IEC 14598) [19]
- Software Measurement standards
    - ISO/IEC 15939 [21]
    - ISO 19761 COSMIC-FFP [22]
- Software Life Cycle Processes standards
    - ISO 12207 : 1995 (plus amendments 2002) [23]
    - ISO 15288 : 2002 [24]

On the level of *tools and methods* the possible support is reduced to a few proprietary products remaining either in stage of a prototype or being under reconstruction. Despite of their actual status these tools demonstrate a vivid interest of academia and industry centers in developing non-manual means of evaluating the quality of software.

These tools are briefly presented below:

- **Software Quality Assessment Exercise (SQAE) –** Robert A. Martin and Lawrence H. Shafer from MITRE Corp USA. April 1996 [25]
    - ➢ Based on quality model built on the combination of Boehm, McCall and Dromey
    - ➢ Evaluates only static quality attributes (4 areas, 7 factors, >100 questions)
    - ➢ Toolkit that is being considered as requiring some modifications [2]
    - ➢ Remaining the subject of the research of graduate students at ÉTS, verifying the possibility to migrate SQAE to ISO/IEC 9126 [3]
- **Multi-layered Customizable Software Quality Model –** Elli Georgiadou from Middlesex University, UK, 2003 [26]
    - ➢ Based on ISO/IEC 9126
    - ➢ Innovative techniques as Composite Features Diagram and Kiviat Diagram
    - ➢ Profiler tool in development, not available publicly

- **QEST Evaluation Toolkit** – Alain Abran from École de Technologie Supérieure, Montréal, Canada and Martin Kunz, Otto-von-Guericke University Magdeburg, Germany. 2003 [27]
    - Based entirely on ISO/IEC 9126 with very good graphical interpretation
    - Non-profit, not for sale
    - For researchers primarily
    - Accessible anytime-anywhere (Web-Internet)
    - Core part only

The theme of the *experience* support to an SQE teacher leaves a bit of a bitter *arrière-goût* (in French – after taste) on the lips. The community of SQE researchers and teachers makes a huge intellectual power with great experience gathered over the years, but:

- usually unpublished or published only fractionally
- kept as notes or course materials
- too often not shared or exchanged in a larger scale
- unevenly distributed (droplet-type knowledge)
- not synchronized

In effect of the above walkthrough of the teaching support to SQE a few conclusions to this chapter could be proposed:

- Teaching SQE is still the role for a "hero", however not without some support.
- Software Quality Engineering may require its own Body of Knowledge named for example SQEBOK (the author reserves for himself the title of the ownership of this name) to advance the maturation process,
- As there is no SQEBOK, can *we*, SQE researchers and teachers become a kernel of this project?

5. What and how the Software Quality Engineer could be educated

The response to this question is hidden in the word "engineering". A Software Quality engineer, as any other *engineer* is supposed to leave in certain moment his alma mater and go to the industry to help it improve and evolve. In the light of such a hypothesis an SQ engineer's education should allow him to implement, measure, evaluate and improve the quality of software throughout its entire lifecycle. Such a vast knowledge requires a structured approach that allows for systematical building the acquired experience that remains, if possible, in accord with industrial practices. A possible structure could consist of, but not be limited to three major (core) components (this proposition focuses only on core areas of knowledge, for example: in Quality Requirements component the techniques of requirements elicitation are taken as prerequisites):

- Quality requirements
- Quality measurement and evaluation instruments
- In-lifecycle quality implementation

Knowledge acquired in *Quality requirements* component should primarily allow an SQ engineer to extract quality requirements from high-level stakeholder's requirements and then to decompose them into lower level categories of quality requirements down to corresponding quality measures (Figure 2) [28].

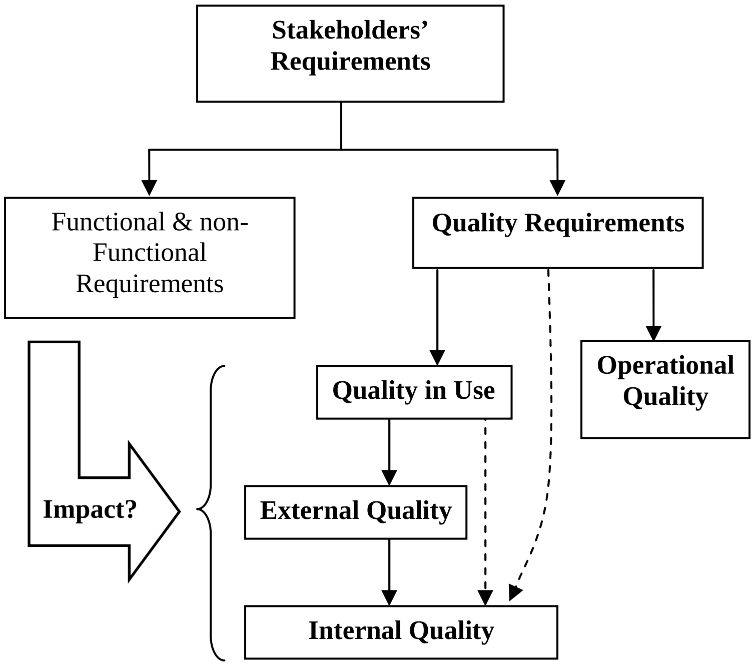


Figure 2. Quality requirements decomposition model (Suryn et al [19])

The proposed decomposition model is based on quality model from ISO/IEC 9126 in conjunction with TL 9000 standard [15, 29, 30] where "the requirements of Quality in Use contribute to specifying External Quality requirements, which in turn contribute to specifying Internal Quality requirements" [28]. The model is well documented so relatively easy to learn and use, but *static*. For a proficient SQ engineer it is important to know not only *what* but also *how*. This approach is addressed by the model for practical process of defining and controlling quality requirements from Fig.3.

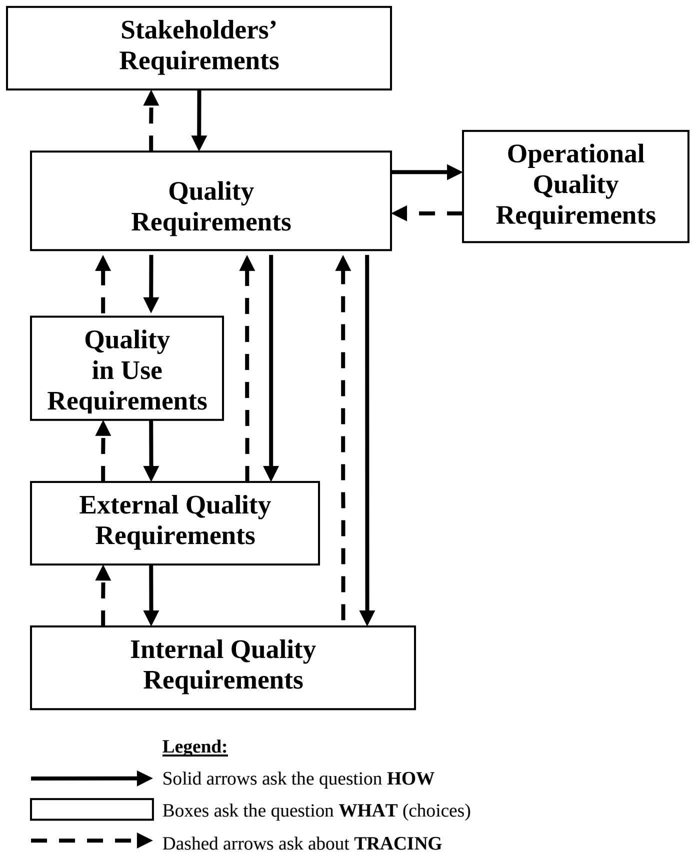

Stakeholders' Requirements
Quality Requirements
Operational Quality Requirements
Quality in Use Requirements
External Quality Requirements
Internal Quality Requirements
Legend:
Solid arrows ask the question HOW
Boxes ask the question WHAT (choices)
Dashed arrows ask about TRACING

Figure 3. Practical process of defining and controlling quality requirements (Suryn et al. [19])

In this model solid arrows indicate the path of questions "how" that have to be asked when executing requirements decomposition, boxes ask questions "(into) what" to help define requirements that result from decomposition process and dashed arrows indicate the path of the subject of traceability of requirements.

The theoretical and practical knowledge of responses to the above questions should allow the SQ engineer to professionally manage the process of quality requirements analysis and definition.

The *Quality measurement and evaluation instruments* component has as the objective to educate the SQ engineer in existing quality models and measures that allow measuring chosen quality attributes of software, and in processes that support this activity. Again, these two elements are well documented and can be easily taught, but also are *static*. The engineering part of the knowledge, which is mapping of these instruments into software lifecycle phases (Fig.4), is much less documented and requires further research. Such a situation may however turn more promising from educational process point of view, as usually the knowledge acquired actively stays longer than this acquired through lectures.

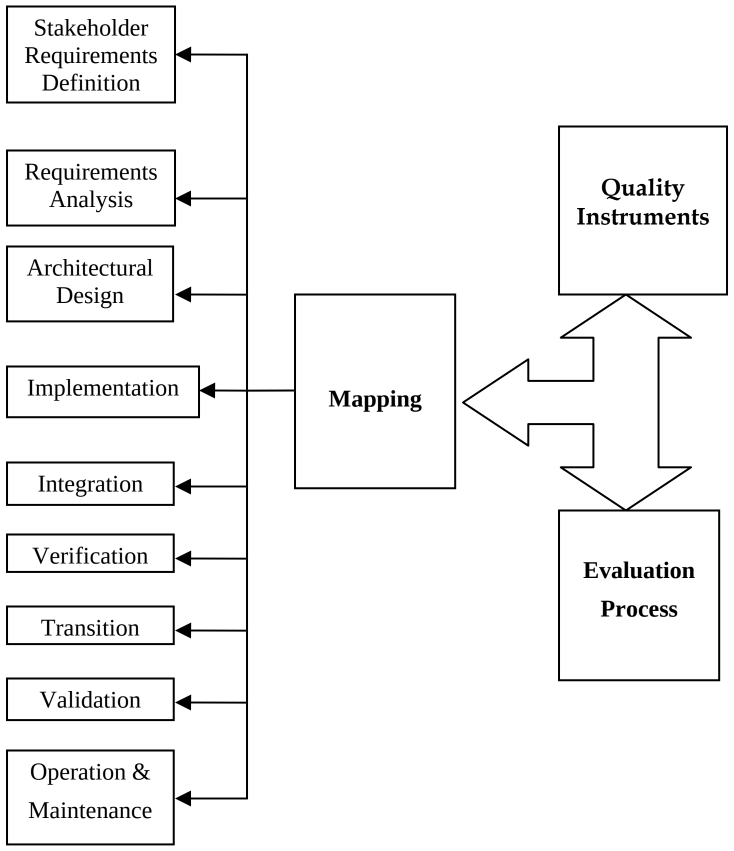


Figure 4. Mapping software quality measurement and evaluation to phases of software lifecycle (Suryn et al. [19])

The last component, *In-lifecycle quality implementation* results from two previous components where the basic knowledge has been acquired. The *implementation* now requires the practical application of this knowledge in course of developing the software (Fig.5).

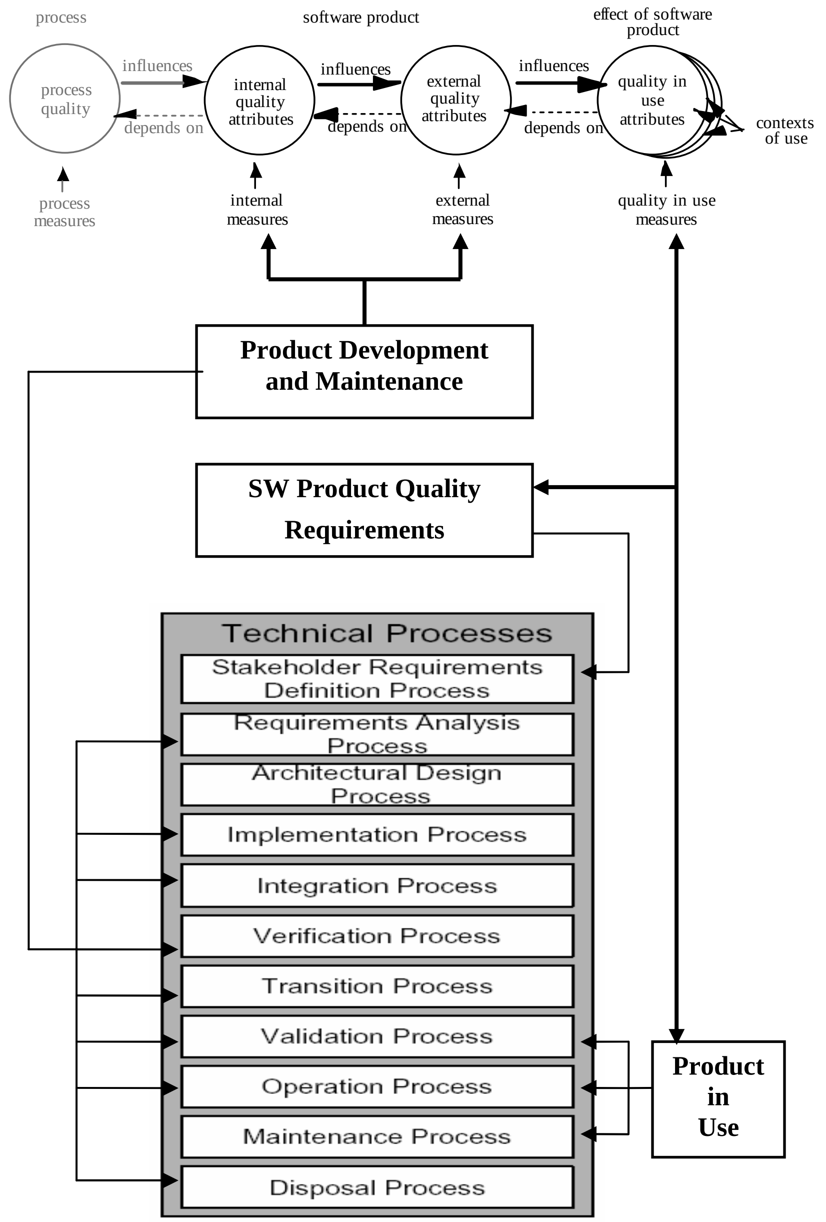


Figure 5. Quality implementation challenge model (Suryn et al. [19])

The education process in this component should allow the SQ engineer to accept the responsibility in the industrial environment not only to identify the correspondence between software lifecycle and quality lifecycle or to define and control quality requirements but first of all to know how to implement, measure, evaluate and improve the quality in all phases of software lifecycle. This role requires and interactive co-operation with development teams and engineers.

But this part is still emerging and requires more data coming from industry and more research affected by the academia.

6. Conclusion

Software Quality Engineering enters now its very active phase of the research, creating *best practices* and industry application. The academic teachers posses enormous knowledge, experience and will to discover new areas and pass it all to students who will convey the new science and best practices to the industry. There are new projects in the domain of Software Engineering body of knowledge and education together with works of International Organization for Standards (ISO) that offer a considerable support to teachers. All that indicates a very promising future for Software Quality Engineering, but to achieve the maturity of this domain both the academia and industry have to invest more in research in many incoming years.

References

1. Abreu E., M., “Consumer groups target software”. *Reuters*, June 14, 2002
2. Halde M., Martin R., Beauchemin P., “Évaluation de la qualité d'un logiciel à l'aide de la méthode SQAE de MITRE”. *École de technologie supérieure*, 2003
3. Côté M-A., Ktata O., Azzouz R., “Migration et Adaptation du Software Quality Assessment Exercise de Mitre Corp”. *École de technologie supérieure*, 2003
4. Shaw M. “We Can Teach Software Better”. *Computing Research News*, 4, 4 September 1992 (pp. 2, 3, 4, 12)
5. Bagert D., J., et al. “Guidelines for Software Engineering Education” Version 1.0. Technical report cmu/sei-99-tr-032 esc-tr-99-002. SEI October 1999
6. Sobel A., E., K., “Computing Curricula - Software Engineering Volume”. Software Engineering Education Knowledge (SEEK), First Draft. *IEEE Computer Society and ACM*. August 28, 2002
7. Sobel A., E., K., “Computing Curricula - Software Engineering Volume”. Software Engineering Education Knowledge (SEEK), Second Draft. *IEEE Computer Society and ACM*. December 6, 2002

8. Vliet H. van, Suryn W. et al. “Thoughts on Software Engineering Knowledge, and how to organize it”. *IEEE Computer Society*, 2003. Prepared for May 2003.
9. Abran, A., Moore, J.W., Bourque, P., Dupuis, R., Tripp, L. “Guide to the Software Engineering Body of Knowledge – Trial Version”. IEEE Computer Society, Los Alamos, 2001
10. Frailey D. et al. “Using SWEBOK for Education Programs in Industry and Academia”. *Southern Methodist University and Raytheon Company*, 2002
11. Suryn W. et al. “Amélioration de la structure du programme baccalauréat en Genie Logiciel”. ”. *École de technologie supérieure*, September 2002
12. Suryn W.,, Robert F., Abran A., Bourque P., Champagne R., “ Experimental Support Analysis of the Software Construction Knowledge Area in the SWEBOK Guide (Trial Version 1.0)”. *IEEE Computer Society*, 2003. Prepared for May 2003.
13. Palza E. “Analyse du contenu du chapitre « Requirements » dans SWEBOK Version 0,95”. *École de technologie supérieure*, Decemeber 2001.
14. Vliet H. van., Software Engineering, Principles and Practice , Second Edition. *Wiley & Son,s* 2001.
15. ISO/IEC 9126 – Software Engineering – Product Quality - Part 1: Quality Model. 2001
16. ISO/IEC 9126 – Software Engineering – Product Quality - Part 2: External Metrics. Planned for 2003.
17. ISO/IEC 9126 – Software Engineering – Product Quality - Part 3: Internal Metrics. Planned for 2003.
18. ISO/IEC 9126 – Software Engineering – Product Quality - Part 4: Quality in Use Metrics. Planned for 2003.
19. Suryn W., Abran A., “ISO/IEC SQuaRE. The second generation of standards for software product quality”. Submitted to *QSIC 2003 – Third International Conference on Quality Software Beijing, China*, September 25-26, 2003
20. ISO/IEC 14598 Information Technology - Software Product Evaluation, Parts 1-6. 1999-2001
21. ISO/IEC 15939 - Software Engineering - Software Measurement Process. 2002
22. ISO/IEC 19761 Software Engineering – COSMIC FFP: A functional sizing method, 2003
23. ISO 12207 - Information technology - Software Engineering - Software Life-Cycle Processes, 1995
24. ISO/IEC 15288 - Information Technology - Life Cycle Management - System Life Cycle Processes. 2002
25. Martin R., A., Morrison S., A., Shafer L., H., “Providing a Framework for Effective Software Quality Assessment: First Step in Automating Assessments”. *The MITRE Corporation*, April 1996

26. Georgiadou E. et al. “Towards a Multi-layered Customizable Software Quality Model”. Proceedings of SQM2003
27. Kunz M. “The prototypical web-based implementation of the QEST model”. *Otto-von-Guericke University Magdeburg, Germany*. 2003
28. Suryn W. et al. “Software Product Quality Practices Quality Measurement and Evaluation using TL9000 and ISO/IEC 9126”. *IEEE Computer Society*, 2003. Prepared for May 2003.
29. TL9000 Quality Management System Requirements Handbook, Release 3.0, QuEST Forum 2001
30. TL9000 Quality Management System Measurements Handbook, Release 3.0, QuEST Forum 2001