Integrating Total Quality Management with Software Engineering Education

Gordon W. Skelton
Department of Computer Science
Belhaven College
1500 Peachtree St.
Jackson, MS 39202

Abstract

During the past several years Total Quality Management (TQM) has become a driving force in the service and manufacturing industries. TQM and Total Quality Control (TQC) are very applicable to the software development industry and thereby, applicable to software engineering instruction. This article focus on the need for integration of TQM concepts into the regular computer curriculum of the computer science and software engineering program.

Introduction

Quality has been defined by both practitioners and researchers to mean "fitness for use" [JURAN] and meeting the requirements as stated in the design phase [CROSBY]. Other definitions focus on the utility of the product or service. In the software arena related definitions have been applied when software is tested against the system requirements, as well as, the reliability of the software and its ability to be maintained. Further verification of the quality of software would include the measurement of the software against the needs of the user/customer.

The software development life cycle, as taught in undergraduate programming and software engineering courses, has over the years focused on analysis of the problem, establishment of the system requirements, design of the solution, coding and debugging, testing, documentation, and implementation. Ongoing maintenance causes this life cycle to be continuous.

Total Quality Management, as applied to the software development process, would alter this waterfall approach. With the use of TQM, various stages of the software development cycle would be reordered. The entire area of quality control and management emphasizes the need to place more effort on the analysis and design phases. With the older development technologies software could be developed with flaws or weaknesses in the design. The later in the development cycle that flaws are identified the more costly it is to repair them. In certain cases the flaws are so severe that the software is rendered useless.

TQM places a high value on not fixing known problems but eliminating them during the analysis and design phases. It is best to build quality in, not attempt to add it later. As in the case of a manufacturing plant, it is quite costly to wait until the inspection process to find a product is defective. Earlier identification of the problem in the design and tooling phase and the elimination of the design flaws can result in greatly reduced defect rates. Dr. W. Edwards Deming strongly believes that the inspection phase of manufacturing should be eventually eliminated. It would be interesting to test this theory in the software engineering process. Moving a portion of the test phase in software engineering to the analysis and design phases can result in a higher degree of software correctness and reliability.

Teaching TQM As A Fundamental Software Engineering Concept

Two distinct models can be used for teaching TQM. As used in both engineering and business administration curriculums, one choice is to include a separate course which introduces the student to the concepts of quality and the implementation of a total quality management program. The second option is to integrate TQM into each of the major courses taught by the computer science department.

The first choice would cause the development of a specific course taught at perhaps the junior or senior level. The course could consist of two parts. The first part consists of lecture and a reading portion which teaches the student the key concepts of TQM and its application to software engineering. A second part of the course would focus on the creation of teams which would implement TQM principles in the analysis and design of a software project. The team role playing would be extremely helpful since team building and employee empowerment are some of the cornerstones of TQM.

The advantage of the second option, integrating TQM modules into the key software engineering courses, allows for the potential application of TQM concepts within a number of different environments. Such reenforcement is quite helpful in the learning process. This integration of TQM into individual coursework requires that each instructor within the department become more than just familiar with the theory and application of TQM. In many departments this knowledge and experience is at best lacking. However, as TQM expands and the work of various researchers and practitioners is published, this lack of knowledge should diminish. The acquisition of "Profound Knowledge", as Dr. W. Edwards Deming describes his theory of quality, is something that must occur over time.

The best situation would be the implementation of both a separate course and the integration of TQM concepts into individual software engineering courses. Using this approach, adequate time could be spent in examining additional areas of the quality movement, quality circles, statistical process control (SPC), human aspects of quality, leadership, and related ideas.

Conclusion

Total Quality Management has a potential for improving the way in which software is developed and maintained. The overall benefits of TQM are far reaching and will allow the student to understand the implementation of TQM in the software industry. Because TQM will be used extensively in the next years in all types of businesses and governmental agencies, students with formal TQM exposure and training will be in great demand.

Overall, I strongly urge that computer science and information science programs begin to investigate the benefits of TQM and related quality control techniques as they relate to software engineeering. Those departments which integrate these techniques into their courses will become leaders in software engineering education.

Regardless of which option is chosen there are benefits that can be gained. Before the levels of success which are expected can be obtained, it is necessary that computer science faculty be trained in TQM and its many facets, particularly as they apply to the software development process. Various organizations are beginning to address this training need. The attached references include a listing of these groups.

References

Avison, D.E. and Fitzgerald, G. "Information systems development: current themes and future directions", Information and Software Technology, 1986, 30, 458-466.

Beizer, Boris. Software System Testing & Quality Assurance. New York: Van Nostrand Reinhold, 1984.

Bollinger, Terry B. and McGowan, Clement. "A Critical Look at Software Capability Evaluations", IEEE Software, July 1991, 25-46.

Card, D. "Understanding process improvement", IEEE Software, 1991, 102-103.

Cho, Chin-Kuei. An Introduction to Software Quality Control. New York: John Wiley & Sons, 1980.

Connors, Danny T. "Software Development Methodologies and Traditional and Modern Information Systems", Software Engineering Notes, Apr 1992, 43-49.

Crosby, Phillip. Quality is Free. New York: New American Library, 1979.

Daughtrey, T. "The Search for software quality", Quality Progress, 21(11), 1988, 29-31.

Deming, W. Edwards. Out of the Crisis. MIT, Center for Advanced Engineering Study, Cambridge, Mass., 1982

Deutsch, Michael and Willis, Ronald. Software Quality Engineering: A Total Technical & Management Approach. New York: Prentiss Hall, 1988.

Flaig, Scott. "Quality and Technology: Sizing Up The U.S.", Solutions, Fall 1992, 4-5.

Gillies, A.C. Software Quality: Theory and Management. New York: Van Nostrand Reinhold, 1992.

Huda, F. and Preston, D. "Kaizen: the application of Japanese Techniques to IT", Software Quality Journal, Vol. 1, No. 1, March 1992, 9-26.

****
INTEGRATING TQ MGT-- continued on page 34

time, broad enough to give them a wider view of the field.

This success, however, has not been without costs. For several years we made Computer Fundamentals prerequisite to CS1. This worked nicely for computer science majors but effectively locked out of our first course, other science and engineering students as well as curious humanities majors. Perhaps even more seriously, making Computer Fundamentals a prerequisite to the first programming course may well have discouraged potential majors. Weren't many of us in the field smitten with the machine as the result of our first programming course? We have, this year, dropped Computer Fundamentals as a prerequisite for CS1 but strongly encourage majors to take in their first or second semester.

The Computer Fundamentals requirement has also posed a problem for sophisticated transfer students, returning students with industry experience, and majors in electrical engineering who decide in their third year that they want to minor in computer science. These students have often picked up most of what is taught in Computer Fundamentals or can easily acquire on their own what they have missed. The solution to this problem is fairly simple. We screen for these students at the beginning of Computer Fundamentals and allow them to fulfill this requirement with a more advanced course.

In the future we hope to write an interpreter for PAL to be run under VMS. Though it is instructive for students to write programs in our pseudo assembler, it is certainly more gratifying to see one's work execute. From the instructor's point of view, a long pencil and paper assembly language program can be difficult to decipher.

We, of course, have a textbook in the works.

Bibliography

1. Brookshear, J. Computer Science: An Overview. Benjamin/Cummings, Redwood City, CA, 1988.

2. Cooper, D. Oh My! Modula-2!. W.W. Norton, New York, NY, 1990.

3. Hutchinson, S., Sawyer, S. Computers: The User Perspective. Irwin, Homewood, IL, 1992.

4. Mandell, S. Computers and Information Processing. West Publishing Company, St. Paul, MI, 1992.

5. Parker, C. Computers and Their Applications. The Dryden Press, Orlando, FL, 1991.

6. Savitch, W. Pascal: An Introduction to the Art and Science of Programming. Benjamin/Cummings, Redwood City, CA, 1991.

7. Tanenbaum, A. Structured Computer Organization. Prentice Hall, Englewood Cliffs, NJ, 1990.

8. Tucker, A., Bradley, B., Cupper, R., Garnick, D. Fundamentals of Computing I. McGraw-Hill, New York, NY, 1992.

10. Tucker, A., Garnick, D. A Breadth-First Introductory Curriculum in Computing. Presented at SIGCSE Conference, February, 1992.

11. Walter, R. Introducing Computer Science with Modula-2. West Publishing Company, St. Paul, MI, 1992.

****
INTEGRATING TQ MGT--continued from page 30

Humphrey, Watts. S., Snyder, Terry R., and Willis, Ronald R. "Software Process Improvement at Hughes Aircraft", IEEE Software, July 1991, 11-23.

Juran, Joseph J. and Gryna, Frank M., eds. Juran's Quality Control Handbook, 4th. New York: McGraw Hill, 1988.

Perry, William E. Effective Methods of EDP Quality Assurance, 3rd Edition. 1987.

Perry, William E. "Quality Concerns in Software Development", Information Systems Management, Vol. 9, No 2, Spring 1992, 48-52.

Shulmeyer, G. Gordon. and McManus, James. Handbook of Software Quality Assurance. New York: Van Nostrand Reinhold, 1992.

Shulmeyer, G. Gordon and McManus, James I. Total Quality Management for Software. New York: Van Nostrand Reinhold, 1992,

Steward. N. "Software error costs", Quality Progress, 1988, 21(11), 48-49.

****
TQ REFERENCES-- continued on page 37

his own pool of "string space" complete with garbage collection. None of the students who chose to develop their own string routines completed a working program by the due date.

4.Next Developments in strings.

Some users of these string implementations are troubled by the "MaxStrLength" limitation: This constant must be known at compile-time and every string will use that much space. This is the standard questions of static vs. dynamic storage allocation. It led to a fourth implementation: StrType.Inf.

In this case, StrType is a pointer to a "block" which resembles the earlier StrType.Std. A block can contain up to "CharsPerBlock" (a new programmer-defined constant) characters and possibly a pointer to another block. Thus, the length of a string is limited only by available memory and short strings can be as short as one block.

Although a working implementation is possible with only the four core routines described in section 2, efficient use of dynamic data structures usually requires some form on initialization. The fifth "core" routine is InitStr (S).

InitStr guarantees that storage is available for S. Every program should call InitStr before using a string. Although the Std, TP, and C versions of InitStr do nothing, the Inf version uses NEW to allocate space for the first block, sets the length field to 0, and sets the first block's "Next" pointer to nil. Separating core operations from higher-level operations allowed creating a reasonably full set of operations on a dynamic data structure with surprisingly little coding.

Certainly other data structures are amenable to this level of separation. I recently shared a layered toolkit for character-based windowing (using these string routines) with a software engineering class (They surprised themselves with the quality of their user interfaces!) and am now re-implementing its "core" to port it to two other compilers running on different operating systems. Maybe "second-degree" data abstraction will help the cause of software portability, too.

I can supply a copy of the "second degree" string routines and some sample programs to anyone who sends one formatted MS-DOS diskette in a self-addressed stamped mailer.

References:

1. Dale, and Lilly. *Pascal Plus Data Structures, Third Ed.* D. C. Heath & Company, Lexington, MA, 1991.

2. Koffman, Eliot B., Stemple, David, and Wardle, Caroline E. "Recommended Curriculum for CS2, 1984," *Communications of the ACM*, 28, 8 (Aug. 1985), pp. 815-818.

3. Kumar, Ashok, and Beidler, John. "Using Generics Modules to Enhance the CS2 Course," *Proceedings of the Twentieth* SIGCSE Technical Symposium on Computer Science Education (Feb. 1989), pp. 61-65).

4. McCracken, Daniel D. *A Second Course in Computer Science With Pascal*. John Wiley & Sons, New York, 1987.

5. Collins, William, and McMillan, Thomas. "Implementing Abstract Data Types in Turbo Pascal," *Proceedings of the Twenty-First SIGCSE Technical Symposium on Computer Science Education* (Feb. 1990), pp. 134-138.

****

TQ REFERENCES-- continued from page 34

Tripp, Leonard L. "Software Engineering Standards: Today and Tommorrow, Part 2. ", Software Quality, September, 1992.

Walton, Mary. The Deming Management Method. New York: The Putnam Publishing Group, 1986.

Training

American Society for Quality Control, P.O. Box 3005, Milwaukee, WI 53201-3005

Specific courses in TQM, Quality Control, Software Quality Assurance.

George Washington University, Continuing Engineering Education Program, 801 22n Street NW, Washington, DC 20052

Offers a wide variety of quality oriented courses.

Juran Institute, Inc. 11 River Road, Wilton, CT.

Special training in quality management as related to software engineering, as well as, courses in quality control and management.

Quality Assurance Institute, Suite 350, 7575 Dr. Phillips Blvd., Orlando, FL. 32819

Provides quality training in a wide variety of information system and software engineering fields. Specific courses offered in testing and measurement.