2005-2548-1119

Software Engineering Standards in the ECET curriculum

Ron Krahe
Penn State Erie, Behrend College

Abstract

This paper introduces the need for including software engineering standards in the ECET (Electrical and Computer Engineering Technology) curriculum today, and discusses the desired depth of coverage. ECET comprises a broad array of topics, including both hardware and software design and development. Many current electrical and computer systems contain embedded controls of one sort or another, and in nearly all of them, the control affects the safety of the user and others, or impacts the efficacy of the system in some way.

An overview of software standards is followed by a listing of national and international regulatory agencies organizations, including both private industry and public government bodies, who have an interest in software engineering standards. The criticality of software has led to the rapid growth in the use of software engineering standards for designers and developers.

After the brief survey, the paper focuses on the comprehensive set of IEEE software engineering standards as an example. And it puts particular emphasis on condensing the full set to a manageable size to be incorporated in an intermediate embedded systems algorithmic processes course.

Introduction

Others [1, 2, 3] have discussed the need for using formal design methods in engineering courses. However, simply using such methods do not particularly facilitate students buying into a complex problem. Experience has shown that it is not uncommon for students to misinterpret an assignment, to solve the wrong problem, to write programs that contain errors and give the incorrect answers, and then blame everything and everyone other than themselves for the mistakes.

This condition is not unique to the education environment. Numerous examples could be given of lengthy product development projects that yielded defective products; products that didn't meet the customer need, and worse yet, programs that performed a miscalculation and caused damage to equipment, and resulted in human injury and death.

Governments, industry, and user organizations have long realized that it is not sufficient to rely simply on the intelligence, cleverness, and integrity of individuals or organizations to produce worthwhile software. Many sets of standards have been written to better control the process of software development.

Copyright (c) 2005, American Society for Engineering Education

By using a standards based approach to teaching software development, students are encouraged to take responsibility for the success of their work, to refrain from belly-aching, to solve a meaningful problem, and to reduce or eliminate errors.

Background

To set the stage for using standards, and to justify in the students' minds the added work necessary to implement the standards, the following ideas are presented:

1. The need for standards:

    a. When there is a failure of safety related software such as that used in medical devices and transportation vehicles and systems, then the results can be catastrophic in terms of human injury, death, or damage to equipment.

    b. When there is any significant product failure, wherein the product fails to meet the customer needs or expectations, then the customer is liable to become dissatisfied and look elsewhere for a more reliable product.

    c. Since most commercial software has an extended life, it will likely go through several modifications to remove hidden errors, alleviate shortcomings, or add improvements in terms of the performance, functionality or ease of use. It must be clearly documented at each point in the life of the software what the software must do.

    d. There is usually a limited budget for software development, and there isn't enough time or money to solve the wrong problem, or fail to meet the customer's needs.

    e. The software will respond somehow to every conceivable variation of inputs. Therefore, the software should be designed to consider all these, and deliver a predictable, safe and effective response. All modes of operation and failure should be consciously considered and addressed.

2. The variety of organizations and standards:

    a. UL[11] (Underwriters Laboratories Inc., USA) is an independent testing and certification agency, and author of over 800 standards related to product safety world-wide.

    b. CSA[12] (Canadian Standards Association, Canada) functions as a neutral third party providing a structure and a forum for standards related to community safety and well-being.

    c. VDE[13] (Association of Electrical, Electronic and Information Technology, Germany) is one of the largest technical and scientific associations in Europe, interested in manufacturing and process automation, transport and medical technology, and transfer of technical knowledge.

Copyright (c) 2005, American Society for Engineering Education

d. CE[14] (European Commission of the European Union) initiated the "New Approach" to standardization, implemented in 1985, resulting in a new legislative procedure at Community level, based on the drafting of essential health and safety requirements and the use of harmonized European standards. EU Directives laid down common technical requirements for each product category and procedures for assessing conformity. National authorities issue certificates of conformity, in accordance with Directives, before products could be placed on the market.

e. ISO[15] (International Standards Organization) is the world's largest developer of technical standards.

f. IEEE[16] (Institute of Electrical and Electronic Engineers) Standards Association (IEEE-SA) is the leading developer of global industry standards in a broad-range of industries, including Power and Energy, Biomedical and Healthcare, Information Technology, Telecommunications, Transportation, Nanotechnology, Information Assurance.

g. FDA[17] (US Food and Drug Administration) provides guidance and oversight of medical device development according good manufacturing practice.

h. NTSB[18] (National Transportation Safety Board) is an independent Federal agency charged by Congress with investigating every civil aviation accident in the United States and significant accidents in the other modes of transportation -- railroad, highway, marine and pipeline -- and issuing safety recommendations aimed at preventing future accidents.

i. FCC[19] (Federal Communication Commission) is an independent United States government agency, directly responsible to Congress, established by the Communications Act of 1934, and charged with regulating interstate and international communications by radio, television, wire, satellite and cable.

3. Topics of standards

   a. Viewing software development as an organized process

   b. Phases of software development and life cycle

   c. Documents related to development

   d. Activities related to development

   e. Constituents and responsibilities

4. Relationship of standards to design methodologies

   a. Functional decomposition and structured methods

   b. Data structured methods

Copyright (c) 2005, American Society for Engineering Education

c. Modeling methods

d. Formal mathematical models method

e. Object oriented methods

f. Ad-hoc methods (just do it, guru hacker attack)

The IEEE has prepared a comprehensive set of standards for developing software. [4] They are widely recognized and extensively used, especially the medical device industry. The standards cover a broad array of topics, much too numerous to cover at one time in one course. So for our purposes, the entire list has been pared down to a more manageable set that still addresses the flavor and intent of using software development standards.

Method

Following is a description of how software standards were introduced in a junior-level intermediate software course on Algorithmic Processes in Engineering. Standards were introduced one at a time with a specific assignment, and were evolved and enhanced in future assignments. The following practical example could be modified to fit other courses.

For purposes of classroom use, the document information flow is reduced to the following:

SPMP	Software Project Management Plan [5]
SVVP	Software Verification and Validation Plan [6]
SRS	Software Requirements Specification [7]
SDD	Software Design Description [8]
SRR	Software Review Report [9]
TPS	Test Procedure Specification (including detailed test procedure) [10]
TL	Test Log
SVVR	Software Verification and Validation Report (including reviews and tests reports)

The *software project management plan* allows the student to review and re-focus on the entire process, including all the various phases of software development. It re-emphasizes the complexity of the process, and the need for a systematic approach. It adds organization to the extended process, and it retards the tendency to sit down at the terminal and begin coding. It includes a commitment on the part of the student to the assignment.

The *software verification and validation plan* continues where the project management plan leaves off, and reminds the student of their role in the success of the project. They are required to buy into the assignment and give serious thought early on to how they will ensure success. For

Copyright (c) 2005, American Society for Engineering Education

each assignment, they can choose the extent to which they will use personal review, peer review, question and answer feedback, and formal testing in the overall development plan. Both of these plans tend to be general in nature, much of the content repeated from one assignment to the next, with variation as needed. Repetition reinforces and expands the concepts.

The *software review reports* are prepared by fellow students at this stage. Having students peer review and comment on each others' plans gives each student added perspective, and improves the quality of their work. Rarely, now, will a student solve the "wrong" problem, or complain that they don't know what to do or how to do it.

The *software requirements spec* includes a restatement of the vague problem, and begins to add specific definition. It limits the scope of the problem, and also clarifies and removes some of the vagueness. It is the beginning of the dialog between the "customer" (the teacher), and the "designer" (the student). The student is given as much latitude as possible in clearly defining an accomplishable problem. Subtle or tricky aspects of the problem come to light during the analysis phase, and are brought out in the requirements specification. This step in the process gives more advanced students an opportunity to use their numerous talents, and also allows more limited students to define a do-able problem for themselves. It encourages each student to think about the problem early, and it is a checkpoint for the teacher, to be sure students are not waiting until the last minute. By thinking about and writing down how the program will be validated often helps clarify an otherwise confusing aspect of the problem. It permits each student to put their arms around their own problem. The software review reports at this stage are prepared by the teacher.

The *software design description* is also written before coding begins. The student gives thought to the operator interface, the class and object divisions, and to the algorithms and language constructs that will be used. The data storage requirements are defined, and the algorithmic approach is described in general terms. Major procedures are described. This detailed design effort sometimes brings to light the need to further clarify or limit the problem, and so the requirements specification will be revised.

The *test procedure specification* is prepared in parallel with the design description, when all the intricacies of the algorithm, ranges of data, and nuances of the operator interface are fresh in mind. It should include a general list of the test cases. Tests for each individual procedure are listed, as well as functional groupings. The detailed test procedure is derived from this specification, and will often be written after the coding has begun or is completed.

Informal testing, sometimes call development testing, it carried out continuously as the code is written, run, and debugged. Errors detected by the editor and compiler are corrected immediately. Repeated runs allow refinement of the operator interface. Detailed tests are developed that will eventually be used for verification and validation.

Once the student feels the program is complete, a permanent copy is made and filed. No more changes are made to the program at this stage. An itemized tabular test procedure is written with specific test cases and input data, as well as the expected (correct) results. Columns are included for actual data, as well as a pass/fail decision for each item. This actual procedure, once run and filled out, becomes the *test log*. This test log is joined to the program. If the student is not

Copyright (c) 2005, American Society for Engineering Education

satisfied with the test results, then the program can be modified and the complete test rerun. This process is repeated as many times as necessary. There is no penalty for the number of revisions, other than the obvious time spent by the student.

The final project report includes a final statement of compliance (or not) with the original problem. It summarizes the pertinent comments of the various reviews and the results of the formal tests. It also contains the complete *design history file*, including all revisions of all specifications and documents, as well as all formally tested versions of the program, full review comments by the teacher and other students, and the detailed test log. The final report is usually submitted on-line in a compressed file, parts of which that can be accesses and referenced as required.

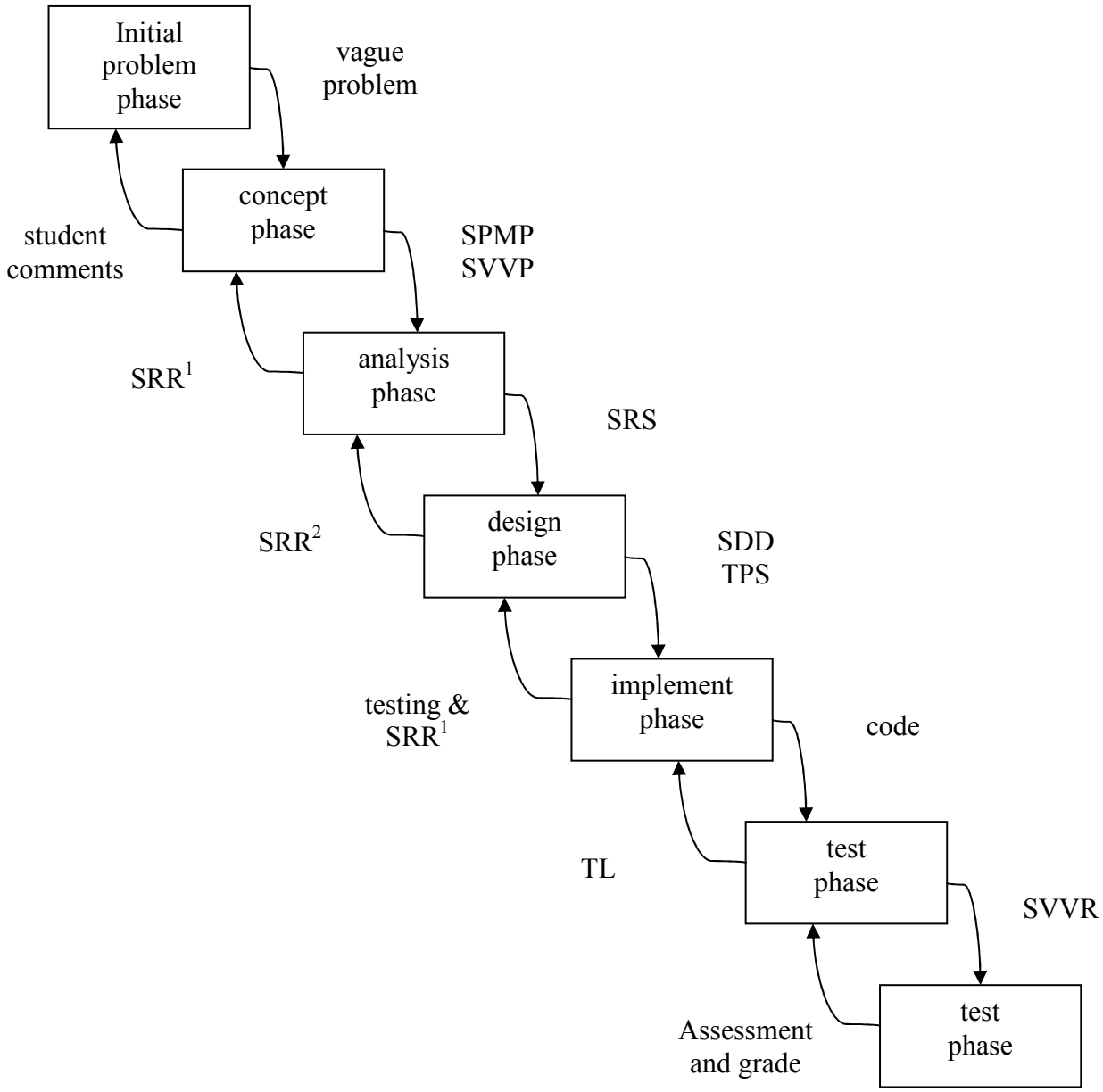
The work product data flow is shown pictorially in Figure 1.

A new project is assigned every two weeks. The total duration of each project is approximately three weeks, with a little overlap during the beginning and end of each project. This allows time for student discussions, feedback, and written reviews. The initial (vague) problem is presented in class, and there is opportunity for questions and clarification. One week is allowed for writing the management plan, verfication and validation plan, and draft requirements specification, and the related student discussions and receiving written peer student feedback. The revised and finalized requirements specification is submitted for teacher comments. During the two schedule lab periods, students have opportunity to ask further questions and get help as required. Students also spend time outside scheduled class and lab periods to write and test the program. The students have one week after the second scheduled lab period to organize the documentation and complete the final report.

Conclusion

Using this formalized approach to software development based on industry standards has greatly improved the quality of student work. It has significantly reduced the misinterpretation of the assignment. It give students a method of attacking a problem without sitting right down at a terminal to begin coding. It encourages them to think about the problem from various viewpoints. It allows for cross pollination of ideas among students. It exposes students to the method of software development that is widely used in practice today.

Copyright (c) 2005, American Society for Engineering Education



SRR[1] Classroom discussions, student discussions, written student feedback.
SRR[2] Teacher-student discussions, written teacher feedback.

Figure 1 Work product data flow diagram

Copyright (c) 2005, American Society for Engineering Education

References

1. Hankley, William, *Software Engineering Emphasis for Engineering Computing Courses: An open letter to Engineering Educators*, ASEE Proceedings, 2004-2305.
2. Schmuller, Joseph, *UML, 2nd Edition*, SAMS Publishing, 2002.
3. *Software Engineering Method Taxonomy*, http://members.aol.com/kaizensepg/methods.htm
4. Schmidt, Michael E. C., *Implementing the IEEE Software Engineering Standards*, SAMS Publishing, 2000.
5. IEEE Standard for Software Management Plans, 1058-1998.
6. IEEE Standard for Software Validation and Verification Plans, 1012-1998.
7. IEEE Recommended Practice for Software Requirements Specifications, 830-1998.
8. IEEE Recommended Practice for Software Design Descriptions, 1016-1998.
9. IEEE Standard for Software Reviews, 1028-1997.
10. IEEE Standard for Software Test Documentation, 829-1998.
11. UL, http://www.ul.com/info/standard.htm
12. CSA, http://www.csa.ca/standards/
13. VDE, http://www.vde.com/vde_en/
14. CE, http://www.eurunion.org/legislat/standard/standard.htm
15. ISO, http://www.iso.org/iso/en/ISOOnline.frontpage
16. IEEE, http://standards.ieee.org/sa/index.html
17. FDA, http://www.fda.gov/
18. NTSB, http://www.ntsb.gov/
19. FCC, http://www.fcc.gov/

RONALD P. KRAHE

Ron Krahe has over 30 years industrial experience in product design and development related to embedded controls, sensors and instrumentation. He joined Penn State in 1988 and currently holds the rank of Associate Professor of Engineering. His teaching specialties include embedded controls hardware and software design, vision systems, and electricity and electronics. His research interests include mechatronics, embedded controls, and sensors and signal processing.

Copyright (c) 2005, American Society for Engineering Education