Quality Nugget

An Insider View of an IEEE Standards Working Group

By Eva Freund

I have often been asked what it is like to participate in an IEEE standards working group. My answer is always that each group is different and I have enjoyed working on all of them. Each of the working groups had its own culture based on the background and role of the members, the leadership of the chair, and the duration of the life of the group. I have limited my discussion to my experiences working on the P829 (IEEE Standard for Software and System Test Documentation), the P730 (IEEE Standard for Software Quality Assurance Processes), and the P1012 (IEEE Standard for System and Software Verification and Validation) working groups.

With the P829 I was the vice chair and played the role of teacher as I encouraged the chair to move the standard away from being merely a documentation standard and toward the concept of test as a *process*. This was in response to the direction taken by the IEEE standards group and the IEEE Standard for System and Software Verification and Validation. The P829 working group consisted of those who performed tests, managed tests, and taught test and/or computer science courses at the university level. They represented a variety of national and international organizations.

The chairs first formulated an outline, which they circulated to the working group. Based on the comments they received, they modified the outline. The vice chair drafted the text for the test process. The plan included a description of test tasks undertaken throughout the development life cycle, and a description of the methodology for identifying and using integrity levels to guide the breadth and depth of test activities. At the same time, the chair was busy updating the text describing the test documentation, including the new master test plan that described the management of the test process. The draft document was submitted to the entire working group for their comments. The chair contacted the reviewers to discuss the disposition of comments. Following the review, the working group members reviewed the tables and annexes and again received many comments.

At this juncture members of the working group were invited to a face-to-face meeting to review the comments. This small subgroup met a few times and became responsible for creating a version suitable for balloting and resolved comments received during the ballot process. Due to the intense pre-ballot effort, the ballot and reballot comments were minimal and easy to resolve. The P829 standard was approved and issued in 2008.

The P730 working group, consisting primarily of quality assurance providers and managers, provided a very different experience. The first chair was comfortable letting the group identify its own direction and focus. Under the chair's leadership, members of the working group were invited to participate in 10 face-to-face meetings for the purpose of specifying the content and direction of the standard. After these meetings, a small subgroup was formed to expand on the work already completed while the larger working group was asked to review and comment on the multiple iterations of the as-completed draft document. I was invited to participate in the small subgroup.

The subgroup held numerous teleconference meetings, sometimes several in a single month. The subgroup agreed that all identified software quality assurance (SQA) activities would implement IEEE Std 12207™.

Group members drafted text and circulated it to the subgroup members between teleconference meetings, and then discussed it during the teleconferences. This allowed members to get to know each other and provided opportunity for the dialogue required to resolve ambiguities and disagreements. For example, members agreed to include information on integrity levels to guide the user in determining how much SQA was needed, but they did not tie the integrity levels to specific activities or tasks. Members agreed that tasks that did not directly tie to IEEE Std 12207™ (IEEE Systems and Software Engineering-Software Life Cycle Process) would not be included in P730. They agreed that identifying inputs and outputs for each task should be considered by another working group for the next version, but they would identify which SQA tasks began or occurred in specific life-cycle phases.

The P1012 working group was unique, in my experience, because it retained its leadership and a core membership over many years and many versions of the standard. The 1986 version of this standard introduced verification and validation (V&V) as part of the software engineering process and described the contents of the V&V plan. The 1998 version changed the focus to the software V&V process. The 2004 version introduced the concept of system V&V and the concept of an integrity level. The 2012 version included the concept of hardware V&V. The 2012 version also provides "stand-alone" information. There is a section for all that is common, a section that applies only to systems, a section that applies only to hardware, and a section that applies to software. The working group continues to improve the standard, especially the section on hardware, and has recruited hardware specialists for that purpose.

Unlike the P829 and P730 working groups, which saw themselves as coming together for a limited period of time for a single purpose, the P1012 working group identifies as a continuing entity.

My understanding of system and software engineering, and the process of standards development, has been expanded by participating in these three groups and learning from their members and chairs:

- Understanding the different concerns of those who manage versus those who execute
- Overlapping of activities and tasks among test, SQA, and V&V and the conflict of who should do what
- Demonstrating the need for a standard to be general enough to encompass multiple industries and sectors and specific enough to be useful
- Understanding the importance of being able to apply standards to new development methodologies such as agile
- Learning that any documentation is the end result of planning
- Learning that I do not know as much as I think I do, and I can learn more
- Having colleagues and friends I often call upon for advice and assistance

Bring on the next working group. I am ready.

BIOGRAPHY

Eva Freund is president and CEO of The IV&V Group, Inc. She has more than 20 years of verification and validation experience including planning, monitoring, and controlling independent verification and validation efforts for numerous federal agencies. Freund has taught at the George Washington University and Goddard College. She served as vice chair of the IEEE P829 (IEEE Standard for Software and System Test Documentation) working group, an active member of the IEEE P730 (IEEE Standard for Software Quality Assurance Processes) working group, and a long-time member of the IEEE P1012 (IEEE Standard for System and Software Verification and Validation) working group. Freund is an IEEE Certified Software Development Professional and an ASQ Certified Software Quality Engineer (CSQE).

Software Engineering Curricular Guidelines

by Trudy Howles With Commentary by Jorge Díaz-Herrera

INTRODUCTION

A 2007 issue of *Software Quality Professional* featured an article describing the challenges of teaching software quality assurance (SQA) topics in an undergraduate software engineering (SE) program (Laporte, April, and Bencherif 2007). In the article, the authors cited various obstacles, including the cost of materials and tools. Perhaps most important, they wrote that SQA activities appeared to be given a low priority in a typical SE curriculum.

Whether recruiting recent undergrads or welcoming a new team member to a workgroup, what expertise and skills can one expect from a recent graduate with a bachelor of science degree in software engineering? What topics are taught, and how much time is spent on those topics? How much are students exposed to "soft" skills, such as oral and written communication, professionalism, and team dynamics?

Curricular guidelines do exist; at the time of this writing, the current document is Software Engineering 2004 (Díaz-Herrera and Hilburn 2004). SE 2004 covers a breadth of recommended knowledge areas, and includes expected outcomes for evaluation. It is a very detailed document that includes a discussion on how software engineering fits as an engineering discipline, pedagogical

issues, and recommended course content. An update to SE 2004 was recently released for public comment. The update is currently in draft format, and is known as Software Engineering 2013 (SE 2013). It seemed important to summarize the curricular modifications, and examine the new focus areas identified in the revised document.

The summary is followed by a commentary from Dr. Jorge Díaz-Herrera. I invited his comments because he served as a co-editor of SE 2004. He has an impressive background in software. His complete bio appears at the end of this article.

SOFTWARE ENGINEEREING 2004

SE 2004 involved three major efforts:

1. Develop a set of curriculum outcomes including a statement of what software engineering graduates should know
2. Determine and specify the knowledge areas; these are known as the Software Engineering Education Knowledge (SEEK)
3. Define curricular recommendations on how to structure the curriculum

A joint task force representing the Association for Computing Machinery (ACM) and the Institute of Electrical and Electronics Engineers (IEEE) started work on the project in the fall of 2001, and the final version was released in the summer of 2004. Table 1 shows the SEEK areas defined in 2004.

Each of the SEEK knowledge units shown in Table 1 was accompanied with a suggested number of hours of in-class time expected to present the material. An hour was defined as an instructional hour during a typical lecture. However, the authors added that the hours reflected a minimum amount of coverage time, and that the time did not include outside-of-class work time (Díaz-Herrera and Hilburn 2004, 19).

TABLE 1 SEEK knowledge areas defined in SE 2004

Knowledge Area and Knowledge Units (Topics)
Computing essentials: computer science foundations (programming fundamentals, algorithms and data structures, problem-solving techniques, abstraction), construction technologies (API design and use, code reuse, assertions, error handling), construction tools (development environments, GUI builders, unit testing, and profiling tools), formal construction methods (application of abstract machines and specification languages, refinement)
Mathematical and engineering fundamentals: math foundations (relations and sets, graphs and trees, number theory), engineering foundations for software (measurement and metrics, engineering design), engineering economics for software (generating system objectives, evaluating cost-effective solutions)
Professional practice: group dynamics (team and group dynamics, interacting with stakeholders), communication skills specific to SE (reading, writing, oral and written communication), professionalism (accreditation, certification, licensing; codes of ethics and conduct)
Software modeling and analysis: modeling foundations (modeling principles, invariants, mathematical models), types of models (information and behavioral modeling, domain and structure modeling), analysis fundamentals (completeness, correctness, model checking), requirements fundaments (requirements process, managing changing requirements, requirements management), eliciting requirements (elicitation techniques, identifying sources), requirements specification and documentation (software requirements specs, documentation basics), requirements validation (reviews and inspections, validating product quality attributes, acceptance test design)
Software design: design concepts, design strategies, architectural design, human computer interface design, detailed design, design support tools and evaluation
Software evolution: evolution processes, evolution activities
Software process: process concepts, process implementation
Software quality: software quality concepts and culture, software quality standards, software quality processes, process assurance, product assurance
Software management: management concepts, project planning, project personnel and organization, project control, software configuration management

THE REVIEW OF SE 2004

In early 2011, a review team was formed to review SE 2004. The review team was specifically charged to do the following (Ardis et al. 2012, 2):

- Survey principal curriculum stakeholders to collect feedback regarding needed modifications to SE 2004. The review team targeted stakeholders from academia and industry, and used several methods to obtain the feedback (surveys, Web, direct contacts).
- Analyze the feedback and determine what actions should be taken.
- Report the findings to the IEEE-CS Educational Activities Board and the ACM Education Board outlining the proposed revisions, as well as a statement of effort to implement the revisions.

The review team started by collecting data from stakeholders from academia, industry, and government. They created an online survey to determine priorities, better understand how SE 2004 was currently being used, and understand the background of the respondents (Hislop et al. 2013).

The review team completed a respectable outreach effort, reporting respondents from 42 countries that included teachers, researchers, software developers, and administrators. It was also interesting that respondents reported varying years of experience: most reported more than 12 years (56 percent); individuals with less than three years' experience were the next highest occurring (13 percent) (Ardis et al. 2012, 3-5).

Based on the stakeholders' feedback, it was determined that the necessary revisions were relatively minor. The review team's assessment was that fundamental components of SE 2004 were "sound" and did not require extensive changes (Ardis et al. 2012, 7).

There were, however, areas identified regarding the amount of time spent on certain knowledge areas. They reported that more time was needed studying the fundaments of requirements. The survey also exposed technology and culture changes such as the need to incorporate "modern software development methods" including agile methods, and a stronger emphasis on security into the guidelines. They also acknowledged that service-oriented computing is becoming more popular and important, and also needed to be added (Ardis et al. 2012, 7). The actual changes to these topics are noted in the following section.

SOFTWARE ENGINEERING 2013

Based on the review team's report, a project was initiated to update SE 2004. A draft of the revised document, Software Engineering 2013 (SE 2013) was released for public comment in December of 2013. What follows are Díaz-Herrera's comments on the draft:

Díaz-Herrera's Comments

The 2013 Software Engineering curricular guidelines for undergraduate degree programs retained the same volume structure as the original SE 2004 volume. There were a few changes in some of the chapters. Chapters 1, 3, 5, 7, and 8 remained unchanged for the most part. Chapters 2, 4, and 6 had the most substantive changes.

Chapter 2, "The Software Engineering Discipline," introduced a subtle change with an emphasis on considering software engineering a professional discipline. "The term software engineering is not necessarily viewed as a 'professional' activity. One of the concerns for these curriculum guidelines is to help with the evolution of software engineering towards a more 'professional' status." This is a welcome change; however, the revision did not go far enough. Although it is telling that the discussion of (traditional) "engineering design" was removed from this revision, it is not clear what is meant by "professional status."

"Could there be a basis for understanding the complexity of software such that we can 'engineer' it to have predictable quality and behavior?" The answer to this fundamental question is unknown today.

We contend that software engineering can become a "professional discipline" without it being a branch of engineering. Indeed, it has been tempting to say that as software engineering matures, it is becoming an engineering discipline. Today, however, software engineering is not widely considered as such, at least not in the traditional sense. We have explored the definition of software engineering fully (Díaz-Herrera and Freeman Forthcoming) and have come to the conclusion that software is different from much of engineering, although there are some similarities (Díaz-Herrera 2009).

The lack of acceptance of software engineering as a professional discipline probably has to do more with our insistence in retrofitting, from the outset, a fundamentally computing discipline with the traditional "engineering" paradigm. In this way, we view computing as a discipline on its own right, with software engineering part of it, and separate from engineering or science, but with elements of both.

Chapter 4, "Overview of Software Engineering Education Knowledge," understandably experienced the most substantive changes from the original 2004 version. It did reorganize one knowledge area, splitting it into two areas; two knowledge areas were deleted, merging the content elsewhere; another area was deleted entirely; and a completely new knowledge area added. In this way:

- *Requirements analysis and specification* became a new knowledge area with the corresponding knowledge units taken out of the existing software modeling and analysis knowledge area. Given the importance of requirements, this change is justifiable,

elevating it from a knowledge unit to a knowledge area level.

- The knowledge area *software evolution* was removed and its topics incorporated into the software process knowledge area. Evolution fundamentally addresses "maintenance" issues and, since changeability is one of the essential problems in software, this change diminishes its importance.
- The knowledge area *software management* was also removed and its topics incorporated into the *software process knowledge* area. Merging software management issues into process blurs the important distinction between project and process control (McDonald Forthcoming). The former refers to "overall management control of entire software development projects." The latter is more focused on process control of specific approaches to development that can be applied to portions of a software project (e.g., coding, inspections, testing). Software development projects must be planned, organized, monitored, and controlled by their project manager. Software development processes define specific technical activities to develop a product. "Controlling a project and controlling the underlying processes are very different activities, and consequently, they require very different techniques to make them effective." We therefore do not support this change.
- The security knowledge area is new. Identifying a new knowledge area for security may make sense. However, security concerns span the entire life cycle, and what we are really talking about here is "software assurance" (Mead, Shoemaker, and Woody Forthcoming). According to the U.S. Committee on National Security Systems, software assurance is "the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner" (CNSS 2010).

Analysis of the topics in this new knowledge area indicates that it may make sense to distribute these topics across the curriculum—since security concerns affect the entire software development cycle—in corresponding parts such as security requirements, secure design, secure construction, and so on. The unit "Building security into the development cycle" fits in the software process knowledge area, while "Security fundamentals" can be part of the computing essentials knowledge area. Incidentally, as proposed, there seems to be not enough time devoted to the unit "Developing secure software."

- Finally, regarding Chapter 6, "Courses and Course Sequences," much of the material was deleted and instead, the reader is explicitly directed to the SE 2004 guidelines. This is not good practice since the new SE 2013 should be a self-contained document. We recommend that this material be included as an appendix.

CONCLUSIONS

In summary, the new 2013 revision of the software engineering curriculum guidelines represents a modest improvement to the original 2014 guidelines. We strongly recommend taking a stand that software engineering *is* a professional discipline on its own right. We also strongly suggest that the document be self-contained with the aim of replacing the 2004 volume, that is, eliminate any reference to it in terms of content.

One important note is that the documents reflect *guidelines*, and not all SE programs are directly based on the curricular guidelines. Certainly, other factors such as accreditation requirements impact the curriculum and amount of time spent on topics.

REFERENCES

Ardis, M., D. Budgen, G. W. Hislop, R. McCauley, M. J. Sebern. 2012. AC 2012-4645: Revisions to software engineering 2004: Curriculum guidelines for undergraduate degree programs in software engineering. Available at: http://www.asee.org/public/.../8/papers/.../download/SE2004ReviewFinal%20(4).pdf.

CNSS. 2010. Committee on National Security Systems (CNSS). National Information Assurance Glossary: CNSS instruction no. 4009. Available at http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf.

Díaz-Herrera, J. L., and T. B. Hilburn, eds. 2004. Software engineering 2004. Curriculum guidelines for undergraduate degree programs in software engineering. A volume of the computing curricula series. Available at: http://sites.computer.org/ccse/.

Díaz-Herrera, J. L., and P. A. Freeman. Forthcoming. Discipline of software engineering: An overview. In *Handbook of Computer Science and Software Engineering.*

Díaz-Herrera, J. L. 2009. *ACM SIGSOFT software engineering notes* 34, no. 5 (September).

Hislop, G. W., M. Ardis, D. Budgen, M. J. Sebern, J. Offut, and W. Visser. 2013. Revision of the SE 2004 curriculum model. *SIGCSE'13*, March 6-9, Denver, CO.

Laporte, April, and Bencherif. 2007. Teaching software quality assurance in an undergraduate software engineering program. *Software Quality Professional* 9, no. 3:4-10.

McDonald, J. Forthcoming. Project and process control. In *Handbook of Computer Science and Software Engineering.*

Mead, N. R., D. Shoemaker, and C. Woody. Forthcoming. Software assurance. In *Handbook of Computer Science and Software Engineering.*

SE. 2013. Draft for Public Review: Software engineering 2013. Curriculum guidelines for undergraduate degree programs in software engineering: A volume of the computing curricula series. Available at: http://computing-portal.org/sites/default/files/SE2013Draft.pdf.

BIOGRAPHY

Jorge L. Díaz-Herrera is president of Keuka College. He was previously a professor and founding dean of the B. Thomas Golisano College of Computing and Information Sciences at Rochester Institute of Technology in Rochester, NY. Prior to this appointment, he was professor of computer science and department head at SPSU in Atlanta and Yamacraw project coordinator with Georgia Tech. He has had other academic appointments with Carnegie Mellon's Software Engineering Institute, Monmouth University in New Jersey, George Mason University in Virginia, and at SUNY Binghamton in New York.

Díaz-Herrera completed his undergraduate education in Venezuela, and holds both a master's and doctorate in computing studies from Lancaster University in the United Kingdom. He recently completed the graduate certificate in management leadership in education from Harvard University's Graduate School of Education. He also served as a writer for the IEEE-CS Software Engineering Professional Examination, and has global-wide consulting experience.