Kudos to Bob Glass and Rebecca Wirfs-Brock

hat a sad day—to read the November/December '09 issue and see the final columns from two of my favorite authors: Rebecca Wirfs-Brock’s Design column and Robert Glass’s Loyal Opposition.

I have truly enjoyed Rebecca’s insights. She brought a compelling blend of experience and openness that made reading each month an enjoyable journey for me. Even though I’m no longer a designer of programs, I always found some useful take-away. I’m sure we will continue to see her contributions.

It was really tough to see Bob’s goodbye. I’ve been reading his different points of view for decades. I have so many favorite stories that I have shared with my students and colleagues over the years. I will find Software a little smaller, a little emptier without his voice at the closing of each issue. He will be missed.

Linda Rising
independent consultant
linda@lindarising.org

The Role of Systems Architecture

I found Frank Buschmann’s article “Scoping and Requirement Woes” (November/December ‘09) to be completely correct except for two implicit omissions. I talk about the first omission in a separate letter published in the December issue of Computer.

The other shortcoming is Buschmann’s reluctance to explicitly identify systems architecture as the key step before software architects take over their part of the system instantiation.

Systems architecture will constrain not only the software architecture but also the hardware, security, communications, and information architecture, as well as all the other aspects that make up a system. It’s fundamental to systems theory that you cannot optimize a system (from the system’s perspective) by optimizing the subsystems (as seen from the subsystems viewpoint). That’s why all the areas comprising a system need to be coordinated as a whole and software cowboys need to be reined in.

Buschmann properly notes the problems with system scope but says, “These mistakes aren’t the prime responsibility of architects ….” I suspect he was referring to software architects, because these scoping issues are the purview of systems architects.

Software folks have always been reluctant to accept that they’re just a part of the entire system. This misleads them into doing silly things that promote failure, both for them and the system. This isn’t to disparage their defensive actions to avoid problems, but they still need to learn to play nice with the rest of the system.

I cannot say whether it’s due to oversized egos, a propensity to tunnel vision, or other causes, but I’ve seen it in software people for over 45 years. Software has been lost much longer than Moses who only wandered in the desert for 40 years.

I’d like to see an entire issue dedicated to the role of systems architecture and where software fits as a supporting element, not the raison d’etre. By all means give the opposition some space to show why they don’t need such an approach. Maybe we can gain insight on where agile methods stop being useful.

In my experience, a dynamic programming approach starting with the systems architecture and using a helical model (which is similar to but distinct from a waterfall model) is a correct way to instantiate proper solutions of typical real-world problems. It does so by handling issues such as Buschmann notes. It can be tailored down to fit problems of any size and can even be omitted for very small ones. (See an example in my letter to the editor of previous issue of Software).

Are there other proper approaches? Nobody has shown us one yet.

In 45 years, I’ve seen software methodologies come and go, but none has established a proper
Defining and Using Domain-Specific Languages

In “A Pedagogical Framework for Domain-Specific Languages” (July/August ’09), Martin Fowler notes that “language workbenches are still in their early days, but if their potential is realized, they could change the face of programming.” We believe the latter part of the sentence is possibly correct, but the first part is false, as it’s some 40 years late. Forty-two years ago, Daniel Teichrow established the ISDOS (Information System Design and Optimization System) project. Its goal was to create software that could translate a target domain’s high-level “problem” statements into a computer application that specified both the solution’s architecture as well as the programs required to execute it (D. Teichrow and E. Hershey III, “PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems,” IEEE Trans. on Software Eng., January 1977). This software was already a full-blown “language workbench,” providing users with a problem (domain) definition language, an analyzer of this language, and a system optimization and design algorithm to generate the solution. The creation of the Problem Statement Language (PSL) required abstracting a number of popular specification methods into objects (for example, Process) and relationships (for example, Consumes), becoming in effect the first method(ology) and domain engineering exercise. The Problem Statement Analyzer (PSA) likewise evolved into a set of reporting functions imitating representation formats of multiple methodologies. Subsequent versions of PSL/PSA used an entity-relationship style metamodel and, in the late 70’s a metalanguage was created in which any domain-specific language (DSL) version (including PSL) became an instance. This expanded PSL/PSA to a full language engineering capability, including a generalized analyzer for the metalanguage. Combined, the metalevel, domain-independent language and associated analyzer now allowed anyone to create their own language/analyzer problem domain tool offering everything that Fowler’s “language workbench” of today would provide for. These generic capabilities eventually evolved into a tool that’s now referred to as a CAME (computer-assisted methodology engineering environment). This capability, with a more elaborate underlying metamodel, can finally be found in modern DSL tools such as MetaEdit+, but its underlying concepts (and promise) has remained as it was when the PSL/PSA metamodel was first introduced.

The question we in the software engineering community should ask is, why is domain-based modeling so hard to implement, when we know that it works and it technically can be implemented?

We believe there are valuable lessons to be learned from the 40-plus years of experience defining and using DSLs (including the ISDOS Project and its successors Meta Systems and MetaEdit+) in answering why the adoption and use is difficult. Here are a few obstacles: increasing levels of abstraction, difficulty to maintain and transfer domain models; high early adoption and learning costs, volatility of target environments; lack of programming languages to support adequate abstractions; and demonstrating the long-term value of domain modeling investments. Based on this past experience, we believe it’s unlikely for DSLs to become a general approach in the near term. The verdict, however, is not yet in, and we should focus on true challenges that make this approach difficult to implement.

Kalle Lyytinen
Case Western Reserve University
kalle@case.edu

Richard Welke
Georgia State University
rwelke@ceprin.org

Should We Certify Requirements Analysts?

I’d like to respond to Neil Maiden’s column, “O, Analyst—You’re Barred!” (November/December ’09) regarding the certification of requirements analysts. I don’t work in the requirements field, but I’ve considered whether certification could help the software development profession generally. I suspect so, but not without some changes.

Modern societies usually require competence in activities that could go badly wrong, and if electricians need to be licensed, why not software developers? Before any mandatory certification is introduced, however, I think a couple of conditions must be met:

- There is a real problem to be solved with nontrivial risks and costs that are attributable to a lack of expertise.
- The proposed certification must make a significant, measurable improvement in the problem.

There are several corollaries to these conditions:

- A general result doesn’t justify a particular method. For example, inspections are known to be effective, but that effectiveness depends greatly on the participants’ knowledge and analytical skills. A program teaching a formulaic inspection method to people not otherwise well versed in the field should not automatically qualify.
- The measured improvement must be in the problem itself, and graduates should show a minimum proficiency in performing the task, not simply in answering questions about how it should be done.
- The improvement should be from the average state of practice, not a strawman state of complete unfamiliarity with the task. If certification is to be mandatory, it should make a difference in the real world.
- The skills taught and tested for should be chosen because of the difference they make, not because of their being teachable and testable. If that can’t be done, certification isn’t the answer.

Nothing in this list requires a certification program to cover a whole field; measurable improvement in a useful subfield is justification enough for it to be considered in staffing, although if a case for mandatory certification is based on a broad field, candidate programs should be measured by their improvement in the field as a whole. Among the invalid reasons for certification: a desire to appear skilled other than by being so; for an organization to look like a center of excellence, unless it is; to create a closed, guild-like profession; or to make...
it easier for nontechnical human resources staff to screen technical candidates.

While these principles, rigorously applied, should prevent dumbing-down, it will remain a risk, not least because these criteria aren’t likely to be applied to optional schemes. I’m also skeptical of schemes promoted by businesses that can expect to increase their business from their adoption. While Martin Glinz’s condition of no monopolies would help, tacit collusion is still possible. I would prefer decisions to be made by entities with an interest in and understanding of the field but with no stake in a particular implementation, such as the engineering institutions.

Neil’s article appeared alongside one about formal methods, a field that seems to be undergoing something of a renaissance for mission-critical development. This makes me wonder if the proponents of requirements certification are ready to take the lead here—for example, by requiring an ability in X or related formal modeling (proof of these methods’ effectiveness is incomplete, but that’s not such an issue for nonmandatory certification). Doing so would go a long way in convincing me that there’s more to certification than just setting a lower bound on competence.

Andrew Raybould
Thales Fund Management
andrew.raybould@gmail.com

Reading around the Edges
Thanks to Philippe Kruchten on his Career Development article “You Are What You Read” in the March/April 2009 issue.

I too have noticed that reading is out of style and I was beginning to wonder if I was the odd one out. It’s refreshing to find my views supported. Oh, and if you ever need another reference about the importance of reading, take a look at Andy Hunt and Dave Thomas’s The Pragmatic Programmer.

My own personal peeve is that so much focus these days is on immediate task-oriented reading. Google is, of course, the extreme of this. Rather than getting a decent book on a subject, programmers are being pushed toward googling for the latest hint. However, even if people do buy books, they tend to do so in a hurry to address a particular technical complication. I admit to doing it myself occasionally (I never have read more than half of Don Box’s Essential COM). However, I believe in reading general books around the edges. My latest is Hunt’s Pragmatic Thinking and Learning.

I find that reading around the edges helps in suggesting new ideas and prevents me focusing too closely and not seeing the wood for the trees. I am sure my reading of Michael Feathers’ Working Effectively with Legacy Code has helped me lead our team to better methods.

In the days of yore, we were expected to read the classics. I really must do that one day—Plutarch in particular, judging from the references I occasionally see. However, I find it disconcerting when so many have not read the classics of software development—Fred Brooks, Ed Yourdon, Jon Bentley, Don Norman, Steve McConnell.

Bill Medland
Sage
bill.medland@sage.com

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.

Call for Articles
IEEE Software seeks practical, readable articles that will appeal to experts and nonexperts alike. The magazine aims to deliver reliable information to software developers and managers to help them stay on top of rapid technology change.

Submissions must be original and no more than 5,400 words, including 200 words for each table and figure.

Author guidelines: www.computer.org/software/author.htm
Further details: software@computer.org
www.computer.org/software