JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 3, May-June 2003

Computational Diversity, Practice and a Passion for Applications

Thoughts On Software Education

Dave Thomas,
Bedarra Corp., Carleton University and University of Queensland

THOUGHTS ON SOFTWARE EDUCATION

During the past few years I've had the privilege to speak on an industrial perspective on academic education. It has been most encouraging to meet numerous academics who are concerned with effective teaching of applied computer science, software engineering, and information technology. In particular I applaud the work of Joseph Bergin and Jutta Eckstein, Pedagogical Patterns Project [1], and those who each year organize the OOPSLA, ECOOP, CSE, PLOP and Agile/XP Educator workshops. Similar groups exist in other communities outside OO, in SIGSOFT, ICFP, etc.

I was surprised that many educators commented that my views were too academic and out of line with real industry. However, several educators resonated with one or more of my remarks. What saddened me is that these educators expressed great dismay that they too were being forced to conform to a false perception of what real industry needs!

This column is dedicated to those who believe that programming and computation is an intellectually significant and challenging career that demands a wide spectrum education combined with practice. I want to acknowledge numerous discussions with many colleagues from the OO community as well as the influence of Dick Gabriel's thought provoking Feyerabend Project [2] and his Master of Fine Arts in Software[3]. Finally the arguments of Peter Denning on IT education [4] seem to parallel our concern for development of competency through practice. I believe the concerns addressed by this column, although my own opinion; do merit serious consideration and broader discussion in our community.

Cite this column as follows: Dave Thomas: "Computational Diversity, Practice and a Passion for Applications", in *Journal of Object Technology*, vol. 2, no. 3, May-June 2003, pp. 7-12.
http://www.jot.fm/issues/issue_2003_05/column1

THE FALLACY OF THE "RIGHT" THING

The myopic industrial context that is being imposed on many of our best educators and their students is distressing. There is a misleading assumption that software engineering is simply a matter of knowing how to use the latest "right" technology. Thus many university courses are indistinguishable from short term training programs provided by the instant job training market[i].

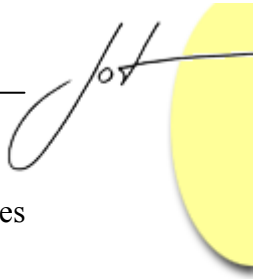
Even more disturbing, the use of the "right" technology is increasingly mandated by funding agencies or university research and development executives who often act as misinformed representatives of the needs of industry. The moment we start believing that there is only one right language, operating system, modeling language, or computational metaphor we are in serious trouble.

It is indeed unfortunate that important 'languages' such as Java, UML and XML, have, through their commercial success, accidentally become "weed languages" that choke out other interesting language varieties that are perceived as less industrially relevant or worse, without utility. Instead of only focusing on the use of the latest "right" technology, a complete education in computer science should include some basic principles which will allow the student to contribute in a future work environment.

PROMOTING COMPUTATIONAL DIVERSITY – OBJECTS ARE NOT EVERYTHING

Students need to see beyond OO and a particular OO technology. Computer Science is not just about objects, just as science is not just about chemistry and business is not just about accounting. Object zealots often have a desire to find overly complex solutions to simple problems. For example, complicated objects and frameworks are used as solutions instead of simpler database and scripting.

Students need an appreciation and understanding of computational diversity. Different tools, techniques, and metaphors are not concepts that should only be offered in optional courses or limited to graduate students. It is not sufficient to make passing reference to this a survey course. Students need to do as many practical exercises as possible. While it is impossible to thoroughly expose students to every concept, their computational journey should include: Table Driven Programming – decision tables, state machines; Symbolic and Functional Programming – Lisp, Scheme, ML, Haskell, O'Caml; Vector Programming – APL, J; String and Tag Languages – ICON, Perl, Omnimark; Rule, Constraint and Logic Programming – Prolog; 4GLs – SQL; Meta Circular Interpreters; AI Programming techniques. In addition to the above, it is important that students understand real machines – registers, caches, instruction sets, interrupts, buses and low level programming. Bread boarding a PIC is a fun and educational way to get ready for the new pervasive computing with sensors and real-time



data handling. Finally, far too many students lack practical database design experiences including ER/OMT, transactions, triggers and non-trivial SQL programming.

COMPETENCE THROUGH PRACTICE

Competence is too often the causalty of curriculum coverage. Students are bombarded with too many concepts and courses. The need to provide breadth, frequently compromises depth in any areas. There is simply insufficient practice to develop mastery. They don't read or write enough programs, and they don't rewrite the ones they have written nearly enough times. Many students have no idea that virtually all good programs have been rewritten 3 or more times. Most programs need more than refactoring, they need to be tossed and substantially rewritten! Programs should get smaller from version to version. Educators should be able to say, look we are just not going to be able to cover the next topic, because most of the students really need to redo the last assignment to demonstrate they actually know what they are doing.

COOPERATIVE EDUCATION CONSIDERED ESSENTIAL

Students need to learn how to work with designs, specifications, and implementations produced by other people. Testing, releasing and refactoring all need to be complete before work is actually "done". Students need an appreciation for domain and technical experience. Effective communication and the ability to listen to others is a requirement. It isn't only the programs that need to get rewritten; students need to polish their written and verbal communication skills.

While software engineering courses make best efforts to provide this sort of educational experience this isn't something that can be done without the substantial support of industry outside the classroom. In our experience the best way by far for students to learn communication and teamwork is through the cooperative education model. In this model, students take 5 years to complete a 4-year honors degree and graduate with two years of work experience. Students alternate academic terms and credit bearing work terms in industry. Many educators are increasingly spending work terms in industry and report that it has had a significant impact on both their teaching and research. Finally the use of cooperative experience in high schools is having a very positive effect on student's attitudes and career choices.

A PASSION FOR APPLICATIONS – COMPUTATION IN CONTEXT

Students need to see objects in context. Students should be provided with exercises based on an actual application that can provide an opportunity to provide a concrete substrate for abstract concepts. Applications by their very nature are open-ended and have extraneous details. Assignments of this type challenge the student to synthesize

knowledge and force interaction with other students, teaching assistants and instructors. The assignment should be open-ended so that it can never be one hundred percent complete. They provide an essential interesting context for learning, especially for non-CS majors.

REQUISITE VARIETY IN CS FACULTY

In the early days of computer science the faculties contained a rich diversity of scientists, engineers, philosophers, and even talented people without a PHD! Each of these brought their external experience context to the department and to the students. Today computer science departments are so pure that if they were a liquid the lighting through them would not be refracted into the beautiful rainbow caused by the impurities. Recently, industry demands for talent have all but removed most of the young faculty with a passion for applications and experimental computer science. Frequently any such faculty are pressed into service to feed the university grant machine or take care of curriculum matters all but abandoned by their colleagues.

The good news is that demographics and economics of the industry that once robbed it of the best applied people now promise to provide an outstanding pool of potential faculty and lecturers who can enrich the educational experience. Even better for administrators, this 50-something population are not seeking tenure nor in many cases full-time employment. Clearly this should not be an excuse to deny new tenure track positions; rather it should be jumped upon as a way to build a modern applied computing education.

THE CHALLENGE OF BEING INDUSTRIALLY CURRENT

Many faculty have moved away from computation simply because they feel incapable of keeping up with what is happening in the industry. Of course they see IT professionals struggle to keep on top of the latest thing. Those of us associated with such innovations need to find ways to keep our educational colleagues current, by helping them understand the essence of each commercial wave without forcing them to crawl through the plethora of quick to market books, tools and APIs. We need to make an effort to show how this is similar but different from what came before, rather than the marketing departments claims of our latest “New New Thing” being new and completely different.

SUMMARY

This is an appeal to those in both industry and academics to work together to refine the practices of computer science education. Unfortunately the excellent IEEE and ACM curriculum for computing education has recently reduced the importance of programming languages, which is a matter of serious concern for educators such as myself. The

challenge for educators is to provide the required breadth of coverage while still providing sufficient depth and experience.

We need to ensure that students, the potential software industry leaders of the future, are exposed to the concept of computational diversity. They need to critically assess ideas and products that are commonly perceived as being "in", while understanding the potential relevance and utility of ideas and products that are "out". A wide spectrum education will give students the knowledge to make technical decisions based on what solution best applies in a given situation.

Computing needs to be taught in an application context, rather than through idealized problems isolated from the messy reality which makes them challenging and interesting. Students need to experience the joy of applications. Finally, best practice needs to be encouraged, even if that best practice means rework and in some cases even sacrificing the breadth of coverage. Work experience programs, which are often called "cooperative education" or "sandwich" programs, should be available at least as an option to all students. Industry needs to help in this process by providing case studies, application examples and real work experience for students and educators.

REFERENCES

[1] Joseph Bergin and Jutta Eckstein, Pedagogical Patterns Project, http://www.pedagogicalpatterns.org/

[2] Richard P. Gabriel, The Feyerabend Project, http://www.dreamsongs.com/Feyerabend/Feyerabend.html

[3] Richard P. Gabriel, Master of Fine Arts in Software, http://www.dreamsongs.com/MFASoftware.html

[4] Peter J. Denning, CACM Columns on IT Profession, http://cne.gmu.edu/pjd/PUBS/CACMcols/

About the author

Dave Thomas is CEO of Bedarra Corp., Adjunct Professor at Carleton University, Canada and University of Queensland, Australia, founding Director of AgileAlliance.com, and founder of Object Technology International. Bedarra works with research labs and commercial partners to transition innovations into products and practices.

[i] In my experience, it isn't possible to consistently build high quality software without well-grounded, bright software professionals. I don't think that everyone who builds applications or works in the industry needs to be a computer scientist, but I would expect them to be as properly qualified in their professional or technical area. In particular I acknowledge that the accidental complexity of many currently available products places great need for and value on technical education - often referred to as *xyz certified*. My concern is that the half-life of such product specific information is very short, and that IT professionals who have only this information are quickly disenfranchised when technology changes. This is a very important topic, but beyond the scope of this column which is focused on academically educated software processionals.

I'm also NOT advocating professional certification, which is another topic on its own.