Combining Theory and Practice to Enhance the Effectiveness and Efficiency of the Software Process

Department of Computer Science, University of Pretoria, Pretoria 0002
Derrick G Kourie
dkourie@cs.up.ac.za

1. Introduction

Although I offered to present a talk on some other research topic at this forum, the organizer invited me to speak instead about a phrase that he had diligently lifted off my web page: "Combining Theory and Practice to Enhance the Efficiency and Effectiveness of the Software Process." His request was based on feedback from last year, indicating that university talks at this forum should try to address "concepts, strategic ideas, plans for the future, etc." The web page phrase, which is now also the title of this talk, was intended to describe a core concern in my personal academic life. Since I will often refer to the title, I will take the liberty of contracting it to the rather capricious acronym CTPEEESP.

Thus, rather than present a specific instance of CTPEEESP – which was my initial intention – my brief is to talk about the title at a meta-level. At first glance, this seemed akin to being asked to talk about the benefits of brushing one's teeth! Why make a high-level case for CTPEEESP when there seems to be universal agreement that it is a good thing? Surely everyone thinks that theory and practice should be combined. And who could be against enhancing the effectiveness and efficiency of the software process?

It is only upon deeper reflection that one becomes aware that there is in fact a whole spectrum of interpretations, intentions, passions and loyalties towards CTPEEESP. Within both industry and academia, people think differently about the matter, even while giving notional ascent to the same words. I therefore propose a categorization of various groupings, firstly within academia and then within industry. Against the background of the political playing field implied by this diversity of mind sets, I will put forward a number of propositions that, I believe, can potentially advance the cause of CTPEEESP.

2. Academic Stereotypes

Before describing a few academic stereotypes, I make two observations. The first is that these are mere caricatures. They may or may not correspond to real people whom you know. Perhaps, like me, some of you will confess to finding little bits of each caricature within yourself. The second is that while I give a computer science perspective, I acknowledge that research into the software process should be multidimensional and multidisciplinary. Many of the human and management sciences can and should contribute to the field.

The Closet Mathematician

The first breed of computer scientist to be considered is best described as a closet mathematician. Highly intelligent and academically successful, the breed's comfort zone is in the world of theorems and proofs, mathematical squiggles and formulae. Its members seek out an international community of likeminded peers and publish voluminously in write-only journals. They are valued within universities for their rate of publication and modest resource needs. All they require to function optimally is an office, a PC, pen and paper. Many within the "formal methods" (FM) community are closet mathematicians. In a recent interchange on the mailing list of FME (Formal

Methods Europe), a young participant characterized members of the breed as "grandfathers" in the following terms:

> "The presentations [at an FM conference] of the old professors left the impression that those guys probably have not programmed more than 100 lines of code in their whole life. They presented, for example, algebraic representations of logic programs and discussed the question [of whether] one syntax should have one semantics or [whether] one syntax may have several semantics for several purposes - very esoteric in my opinion."

However, being secure in a mystical belief that at some indeterminate time in the future the true practical value of their theorems may become apparent, they have no real concern for combining theory and practice. Neither do they have much concern for minimizing effort or maximizing efficiency, being inclined to equate ease of use with superficiality – i.e. unworthy research.

The Niche Researcher

Many academics find their success in a particular research niche in some or other domain: HCI, networking, security, graphics, etc. The research invariably involves the development of prototype systems to test ideas. Often they find financial support for their research from national funding organizations. The best are supported by industry in venture-research undertakings.

But the salient fact about niche researchers is that they are not primarily oriented towards producing *industrial* strength software. Once a prototype has illuminated or verified a research idea, it may be discarded. As a result, the niche researcher generally does not have strong views about what should be done to enhance the effectiveness and efficiency of the software process, (unless, of course, that is the specific research niche). Those who are modest recognize that the *research* software process is inherently different from processes that are appropriate in the software industry. Although they will concede that there is a need for CTPEEESP they will, at best, be supporters of the cause, but not crusaders.

The Gnostic

Closet mathematicians or niche researchers sometimes become so impressed by their own research achievements, that they evolve into Gnostics: the ones upon whom great wisdom has descended from above in all matters related to IT in general, and in relation to the requirements of the software process in particular. Because their academic environment has recognized them as good IT researchers, their associated software must obviously be good. The programming-in-the-large required by industry is seen as nothing more than a whole lot of programming-in-the-small exercises, which is their terrain of familiarity. Industry concerns for requirements solicitation methods, for controlled phased iterative incremental development, for documentation and management procedures, for systematic coding methodologies and standards, for version control, etc – these are all rather tiresome and trivial non-issues. Software engineering is simply what computer scientists and engineers get to do once they graduate. Practical execution of the software process in an effective and efficient manner comes naturally once you have a good grounding in the basics of programming.

The Introverted Voyeur

Standing in contrast to the Gnostic is the young introverted voyeur. Spurred by a brilliant student career, he has been seduced into joining an academic department. His programming-in-the-large experience is limited to the projects assigned to him as a student. Although he has heard of the software crisis, he has little notion of the real issues. Instead, he has been diverted into doing niche research – even doing it well in the eyes of his peers. But he finds himself spending more and more time surreptitiously peering through the windows of his ivory tower into the windows of his industry neighbours. For this sad individual, everything on the other side takes on the hue of being better and more glamorous. While he tinkers in his own little research corner, he imagines that all the important things are really happening out there in industry. They have all the money, they have all the wisdom and they are having all the fun. His inner conviction is that his theory world and the glamorous world of software practice cannot meet. Perhaps one day when he grows up and gains sufficient self-confidence, he might join in the fun on the other side.

The True Believer

Finally, there are the true believers. They have had the good fortune of being exposed to the software process in an industrial context, but have not succumbed to the lure of industry remuneration. Not surprisingly, there are not many in this category. But because they have either worked in industry or acted as academic consultants to industry, they have been forced to trim their sails close to the concerns of industry. They have become keenly aware that the theory that they teach and the things that they research are ultimately only meaningful within the context of some practical software process. They try to inculcate into their students a value system that deeply appreciates and adheres to sound software engineering principles and practice. Their intellectual instincts are to look for bridges between theory and practice. But, as a result of devoting energy to engaging industry, they tend to have less impressive research records than their peers. This weakens their academic credibility and consigns their concern about software engineering issues to a lower priority within academic curricula.

The foregoing was a non-exhaustive but representative set of dispositions within academia towards CTPEEESP. Although there are no doubt local and regional dominances, I suspect that the mix will be found across the globe in most computer science departments, if not in other IT departments as well.

3. IT Business Stereotypes

I can give no more than an outside and tentative view of IT business stereotypes. It seems to me that there are two broad philosophical approaches. Let us call their adherents the cavalier profiteers and the cautious paranoids respectively.

The Cavalier Profiteer

The cavalier profiteer sees profit *as an end in itself*. He takes a short-term view of the world. The horizon is a web year. Efficiency is key. Software is to be delivered as rapidly as possible, even at the risk of severely testing customer tolerance. Like Mao Tse Tung, he believes in creating crises to advance productivity. This he does by committing to unrealistic thumb-sucked schedules. He then hires energetic young wiz kids who, in short edit-compile-test cycles, furiously engage in heroic *ad hoc* programming, clicking at mice and clacking at keyboards into the early morning hours. Immature products are submitted for beta testing and bug fixes are postponed to the next version. There is no time for academic niceties and speculative thought about the software process. Agile programming (á la Beck, Cockburn and co.) is just fine. Documentation is, at best, a nice-to-have; at worst, a waste of time. With dollar signs in his eyes, he believes that one day, he will strike gold. He will corner and capture the market. He will set his own standards. He will grow up to be like Bill Gates. For now, pragmatism is the name of the game. And then, his team of wiz kid Xtreme programmers unexpectedly leaves for greener pastures. But they do not leave behind any documentation, and another dot bomb explodes!

The Cautious Paranoid

We have it on good authority that only the paranoid survive. Perhaps our intrepid and cavalier profiteer may survive by growing into a cautious paranoid. He comes to believe that the business of business is to remain in business. Profits and efficiencies are no longer viewed as ends in themselves, but as means to the end of surviving. Because he becomes paranoid about sudden staff resignations, he insists on documentation and standards in support of maintainability. Because he fears customer disapproval, he puts a high premium on supplying quality bug-free software, even if that means being a little later to market. Concerned to keep uncertainties about the future to a minimum, he focuses more energy on the earlier phases of the software life cycle. Fearful about losing out to competitors, he takes cognizance of his ISO9001 status, his CMM level, etc.

4. Implications

We started off by observing that most people give notional ascent to the claim that CTPEEESP is a good thing. We now see that there are many different dispositions towards it, both within industry and within academia. In our stereotyped scenario, only the true believers and cautious paranoids practice and promote CTPEEESP with any degree of fervour. The disposition of others varies from indifference to tolerance. For many, it is not a foreground issue. Against this canvas of diverse role players, I offer the following thoughts.

a) The IT graduates represent the focal point at which theory and practice are in fact combined. It is they who take away many little pieces of theoretical knowledge and apply them, perhaps even subconsciously, when they enter industry. But their training rewards them for solving problems. Very little emphasis is placed on the *way in which* the problem is solved. At best, such methodological issues are touched upon within the confines of isolated courses bearing titles such as Software Engineering, Object Oriented Analysis and Design, or Software Development.

b) As a result, not enough IT graduates are imbued with a sense of urgency about the need for methodological rigour, the need for adhering to standards, the need for a disciplined and systematic phased approach to developing software. Few are able to connect the formal method theory taught to them by the closet mathematician, with the software process that they are required to implement in industry. Many inherit the lukewarm disposition of the niche-researcher, or the misplaced know-it-all attitude of their Gnostic teachers. As a result, new employees only adhere to the rigours of a software process because of cautious paranoid management pressure from above ? not from inner conviction.

c) It is one thing to debate whether computer science and software engineering are one and the same or whether they are separate disciplines belonging in two different academic departments. It is quite another thing to observe that, *as a matter of fact*, large numbers of IT graduates end up engineering software, irrespective of whether their training was in computer science, software engineering, computer engineering or even electronic engineering. It thus behoves us to inculcate software engineering values and practices into all IT curricula.

d) It is in the interests of those in industry who place a high premium on CTPEEESP to support true believers in academia. The support should not only be financial, but moral as well. They should recognize that true believers are often locked in internal political struggles ? both horizontally with peer academics and vertically with superiors. The horizontal struggle has to do with influencing curricula in the face of indifference, skepticism or downright opposition from colleagues. Vertical battles relate to the tendency of academic authorities to discount the value of practical experience when considering appointments and promotions.

e) It is also in the long term interest of the cautious paranoid not only that the hemorrhaging of IT academics into industry be stemmed, but that it in fact be reversed. In order for CTPEEESP to flourish, universities require a critical mass of IT academics with deep experiential roots in industry. Currently, this is almost impossible, not merely because of the widely acknowledged salary differentiation, but also because of the aforementioned discounting of industrial experience. There are some in industry would seriously consider a switch to academia, despite an inevitable reduction in income. They have the right paper qualifications, the ability and enthusiasm to teach and to do research and they have a wealth of valuable practical experience. But they balk at the reduction in their status from 'being somebody' in industry to being 'a nobody' in academia, merely because they do not have a long list of journal publications.

f) There are many concrete ways in which practical experience can be injected into universities. For example:

   - Industry could facilitate schemes whereby academics take sabbaticals in industry. This seems a relatively cheap and effective way of combining theory and practice. A stumbling block is, of course, the previously mentioned tendency of academic authorities to promote staff on the basis of research metrics rather than experience in the field. High-level industry lobbying to dent this tendency (for example at the CEO-to-University principle level) would, I believe, be both appropriate and effective.
   - There are some in industry who are willing and able to competently present courses at universities. They should be encouraged by management to do so. They should be given the necessary time off, not only to present the lectures, but to design the course, to prepare the classes and, most importantly, to do the necessary assessment. This is very different from the 'evangelical' talks (about the company, its latest products, its assessment of future trends, etc.) that companies are wont to present to university audiences. The former is a long term commitment to transfer values and skills, while the latter is essentially a short-term marketing exercise.
   - Where industry elects to fund universities, it could subtly or explicitly make its donations contingent on things happening that will advance CTPEEESP. For example, a chair in IT could be endowed under the specific condition that its occupant should have appropriate industrial experience.

g) Clearly theory and practice are combined when industry engages academics in specific research projects or discussion forums such as the present. Bringing together the two cultures has an inevitable and mutually beneficial rub-off effect. However, I believe that many opportunities for fruitful interchange and research remain unexploited. In particular, with some notable exceptions in such institutions such as Maryland, there does not seem to be very much collaboration in conducting software engineering experiments. It should be borne in mind that academics have access to a pool of students who can be used in all sorts of experimental trials to test the efficiency and effectiveness of various competing approaches to developing software.

5. Conclusions

This talk could have focused on past, current and future examples of how theory and practice combine to enhance the effectiveness and efficiency of the software process. Instead, I have chosen to focus on a prior concern: conflicting value systems within the academic and business milieu that hinder the desired osmosis between theory and practice. I have suggested a few possible strategies to advance the cause.

I conclude by sharing two contrasting scenarios. The first is of a video-clip I saw recently during a talk by a Microsoft evangelist. Narrated over a sound background of pulsating modern stereo music, the video takes one through a typical day in the Microsoft software development rooms. Angled scenes of trendy twenty-somethings flash by in rapid succession. Rooms are brimful of high-tech equipment; there is animated discussion and furious coding. Everyone is focused on completing the day's code revisions. These are incorporated into a vast software system that does automated builds and tests. By the next morning, a comprehensive bug report is available, which determines the agenda for the next day's activities. The intention of the video clip is clearly to convey a sense of dynamism, energy and dedication to tracing and removing bugs.

In contrast, imagine a quiet room - light and sparsely furnished in minimalist style. Its many windows look out onto a well-kept garden. A simple wooden table and chair is in front of each window and a PC rests upon each table. On the floor opposite each PC, a shaven saffron-robed monk sits silently in a lotus position and contemplates the blank space before him. Then, at mid-day, the monks rise in unison, bow to each other, seat themselves at the PCs and calmly write code according to clearly documented specifications. At the end of the day, they bow and return to their quarters, confident of steady progress towards a targeted software system that contains no bugs.

This last vignette is purely fictional. It comes from a book that exists as a mere fantasy in my head and will probably remain so. A possible title is "Zen and the Art of Software Engineering". It is based on the thesis that the effectiveness and efficiency of a software process is determined by the core values of its participants, most particularly, by their disposition towards bugs. Some regard bugs as an unfortunate but inevitable fact of life: "Everyone makes mistakes. Let's not get too uptight or unrealistic." Such tolerant pragmatists will tend towards a software process similar to the Microsoft scenario: the energy is focused on frenetic *ex post facto* repair. But to the extent that one is idealistically committed to the complete avoidance of bugs and develops a puritanical abhorrence of them – to that extent energy shifts towards thoughtfully, calmly and methodically purging them from the system *ab initio*. The challenge facing the true believer and the cautious paranoid is no less than this: a missionary task to influence the core value systems of key role players in the latter direction.