Conquering the Crisis in Software Engineering

Victoria Livschitz
Sun Microsystems
victoria.livschitz@sun.com

Software engineering today is in a severe crisis. The situation is particularly grim because this crisis is not widely acknowledged by the software development industry or academia. The haphazard sprawl of technology is commonly viewed in our field as an indicator of remarkable progress. However, this jungle in my view has steadily escalated the cost and complexity of industrial software development, and they are now nearly out of control!

The industry hasn't yet heard the wake up call. Thirty years of high-tech evolution governed by Moore's Law produced individual computers and computer networks capable of solving computational problems of massive scale and complexity. Traditional methods of writing computer applications – handcrafted by one or a few individuals using low-level languages and tools – have long become impractical. Commercial software development enterprises have responded with an explosion of new technologies including programing languages, tools, libraries, protocols, middleware, application frameworks, project methodologies, design patterns, reference architectures and so on.

A few well-known characteristics of the present state of affairs will illustrate the consequences. The sprawl of technologies has resulted in over-specialization of developer skills and over-generalization of management skills. Being a “good programmer” no longer guarantees one a decent living – companies want fine-grained expertise in specific flavor-of-the-day tools, products and frameworks. And only a superhuman can manage a technology portfolio while remaining well-grounded in the technical issues. The incredible rate of change leaves professionals constantly scrambling to find “the next big thing,” lest their skills quickly become obsolete. Meanwhile, the costs of development projects continue to rise while the quality of the resulting systems diminishes.

No wonder admissions to computer science and software engineering departments are declining. Why should young people be attracted to software development rather than genetics, engineering, medicine, business or law? Computers no longer carry the aura and cachet of a new frontier. Nor is there the job security which professional developers enjoyed during previous decades. Academia has not adjusted to the sweeping changes in the software development. Programs in software engineering world-wide still focus on the traditional programming-oriented computer science curriculum, and Stanford still teaches the majority of required graduate courses in C!

What can universities do to survive the profound crisis facing software engineering? And how can they help the industry regain a sense of direction on the path of technological innovation?

First and foremost, software engineering must overcome its identity crisis and decide whether it is a branch of engineering, mathematics or computer science. I think software engineering is above all an engineering discipline. Unlike computer science, it does not seek to answer the question, “Can a computer system with specified properties be built in principle?” Not does software engineering attempt to construct a precise model of the problem, a là mathematics. Professional software development is primarily about delivering a product with the desired qualities, on time and within budget. Therefore software engineering must predominantly concern itself with teaching the framework for a rational trade-off analysis between product features, quality and cost. Such a framework is conspicuously absent from the software engineering curriculums.

Secondly, the goal of every software engineering program must be to teach students the science and art of systemic thinking. In my opinion the kernel of the program should be a course in general systems theory, an extended curriculum on software architectures, and a solid grounding in several leading software lifecycle management methodologies.

Finally, current education must be far more practice oriented. Students should be given the opportunity to test the theory and acquire practical software development skills by solving real business problems under realistic pressures. Instead of emphasizing solitary programming, the focus should be on collaborative teamwork.

In closing, it is exceedingly important for the global community of educators to realize the true depth of the crisis that software engineering has been facing. Academia must lead the revitalization of this field which is of immense importance for the global economy, and is in dire need of a new approach. Only then might the software engineering programs at the universities be able to restore the prestige of the software development profession, attract top talent and develop highly effective leaders.