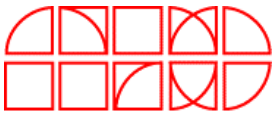

Towards a Professional Software School

Avron Barr and Shirley Tessler
Aldo Ventures, Inc.
www.aldo.com

These notes discuss the need for and sketch the curriculum of a professional software school – like a medical school. Software has become just as important to our economy and to our general well-being as medicine. While traditional computer science and information systems curricula include software as well as hardware, a significant amount of what has been figured out by working professionals about the practice of software development and asset management falls across or between academic disciplines. Furthermore, experiential learning is not easily packaged as a component of computer science, engineering or management programs. What we need is a professional school, with programs that stress realistic practicums and with faculty who include top-flight practitioners as well as scholars and researchers.

The Market: Who Needs to Know about Software

Software has become a major economic force over the last half century: a deciding factor in business strategy and even in warfare, as well as an increasingly important global industry in its own right. As part of the Stanford Computer Industry Project, we conducted a six-year study of software as a global economic phenomenon. We took a broad perspective on software, including:

- Software publishing, both enterprise and consumer products;
- Business and government systems and the affiliated software services sector (which is twice as big as software publishing); and
- Software embedded in devices and machines, from pacemakers to airplanes.

Current offerings for software professionals at most schools target working programmers who need to keep their skills up to date and, less effectively in most cases, engineering project managers and IT managers. There is a huge need and a long-term opportunity in programs targeted at additional careers, including:

- Sales, marketing and other management in the software industry itself, including publishing, services and products with extensive embedded software;
- Decision makers in all industries who regularly make million-dollar decisions about software purchases, projects and services;
- Financial analysts and investors who increasingly see software development and related intellectual property as a key, albeit puzzling, asset;
- Designers of products of all sorts (semi-conductors, telephone systems, consumer electronics, automobiles, home banking, medical instruments, etc.) who increasingly depend on embedded software to achieve functionality, but who receive little formal training in software in most engineering programs;
- Corporate executives, legal analysts and government policy makers, whose decisions often require more than superficial knowledge of software technology and process.

Ideas for the Curriculum

We've sketched here a multi-faceted curriculum for students in different disciplines covering the subject of software, viewed quite broadly. We've tried to be provocative, and we're certainly not complete. Courses in a professional school might include, in addition to lectures and programming projects, prolonged practicums and work-study programs to expose students to real-world techniques and problems.

Computing and Programming Basics. Computer systems and architecture, software technologies, programming languages and methodology, software architecture, platforms.

The Software Industry. The structure of the software industry, global IT business environment, current technologies and trends, software skills and talent, product specification and design, describing functionality, usability, IT as a competitive weapon, software services, marketing products and services, technology partners and distribution partners, sales channels, product and project failures, national software industry development strategies.

Software Development. Requirements specification, engineering design, development planning, hiring and managing software teams, tools, architecture, project methodology and management, code reviews, version control, testability, quality assurance, risk factors, project failures, sourcing software development, offshore outsourcing, project and systems documentation.

Software Project Management. Requirements analysis, systems analysis, the development cycle, organizing and managing software teams, project planning, staffing, monitoring progress, quality assurance, cross-functional project management, metrics, cost controls.

Software Quality Assurance. Designing for testability, testing techniques and tools, bug management, version control, release management.

Enterprise SW Management. Alignment with the business, software as a strategic weapon (e.g., in insurance, banking, manufacturing, high tech, government, ...), measuring productivity gains, arguing ROI, project management, gathering business requirements, development cycle in the enterprise, costing and accounting, enterprise SW architecture/infrastructure/platforms, legacy systems, the CIO's job, organization of an IT department, database administration, current trends (data mining, portals, knowledge management, e-commerce, mobile services, customer relationship management, model-driven architecture, …), buy-vs-build decisions, buying (and selling) enterprise software products and services, software maintenance, vendor and outsourcing relations, litigation, security, offshore development, intellectual property management, issues in integration.

The Software Product Business. Product life cycle, marketing requirements documentation, product design, engineering specifications and plans, platforms, architecture strategy, product management, engineering management, feature lists, technical support, beta programs and reference accounts, technology and marketing alliances, packaging and manufacturing, distribution channels, documentation and training, localization, customization, integration, version management, release management, tech support.

Human Resources. The composition of software teams, recognizing software talent, hiring and managing software teams, skills development, life-long learning, career paths, management of global teams.

Embedded software. Synchronizing HW and SW in product development, prototyping, real-time software and architectures.

Software Law. Intellectual property, patents, contracts and licensing, antitrust, trade secrets, copyright, development and marketing partnerships, performance liability, warranties, litigation.

Other courses. Software entrepreneurism, the software services business, the history of software platforms and programming tools, software marketing, …

Research of a Different Sort

In the same way the Chemical Engineering departments conduct a different kind of research than Chemistry departments, we believe that there are important research issues in applied software science that are not currently supported in other academic disciplines. With funding from the Sloan Foundation, we were able to study some of these issues briefly while at Stanford. Perhaps 3 or 4 professors there, and a few dozen at other schools, do research in related areas. The kind of school we are describing would attract scholars who are interested in studying questions like the following:

- Benchmarking total corporate expenditures on software products, services, development, …
- Software project management: best practices, new tools, …
- Software development and testing tools
- The software industries of other countries (India, Israel, Ireland, Japan, …)
- The economic impact of software failures: projects, systems and devices
- Intellectual property and the software patent situation
- The nature of software talent and global supply and demand
- Financing software R&D and the implications of early-stage acquisition of startups

A Timely Opportunity

We think there is a unique opportunity now to take advantage of Silicon Valley's professionals as adjunct faculty, consulting professors, or guest lecturers. We have seen the strong contribution of these folks in both the engineering and business schools at Stanford. First-class practitioners will be increasingly available in the coming years as a resource for teaching and for curriculum development. They want to contribute, if for no other reason then to codify their life's work. And they know things about the practice of software engineering that are not in textbooks. Working together with existing faculty, they could develop unique courses that would be of extraordinary value to current and aspiring high-tech professionals. The need for this kind of education continues to grow. There is plenty of time to start small, to figure out how to organize and teach this material, and to build a professional program piece by piece.

There's a lot of money in the software industry. Looked at as a business, professional software education could expect considerable funding from the software companies. In addition, government research grants focused on issues in applied software science should be forthcoming, since software is a major cost and risk in their operations, as it is in industry. (The State of California, for instance, has lost many millions of dollars in failed systems efforts.) Finally, we believe that a uniquely positioned program of this sort could generate substantial revenue from international executive education, curriculum royalties, and distance learning offerings.