Education and Social Development of T-Shaped Software Engineers

Barry BOEHM[1,a,*]

[1]University of Southern California (USC), Los Angeles, CA 90089-0781, SAL 328, USA

[a]boehm@usc.edu

[*]Corresponding author

Keywords: Software Engineering, T-Shaped Engineer, MS Curriculum, Real-Client Project Course

Abstract. A T-shaped engineer is one who has deep knowledge and skills in a particular discipline, and also has a working knowledge of other disciplines important to the engineering projects they contribute to. Many computer science departments graduate mostly I-shaped students. They have strong knowledge and skills in computer science (CS), but little knowledge and skills in other disciplines such as economics, management sciences, human factors, or physical sciences important to their organization's projects. This paper summarizes my experiences with a large, diversified company, TRW, in addressing the challenge of broadening new I-shaped CS graduates to become full participants in TRW projects; learning about T-shaped engineers from Simon Ramo (the R in TRW); and deciding to early-retire from TRW to create a software engineering curriculum at USC that would produce more T-shaped MS graduates, and ultimately produce more T-shaped BS graduates.

The paper proceeds to describe the resulting MS degree in CS, with specialization in Software Engineering, and particularly its real-client project course, in which teams of 6 students develop a software system satisfying the needs of a USC campus or neighborhood organization. This addresses some other shortfalls in undergraduate computer science students' education, in which they perform and are evaluated as individuals vs. being effective team members, learning about pure computer science topics such as software algorithms, data structures, programming languages, compilers, operating systems, database management systems, programming, software testing and debugging. The team project course also exposes them to parts of a project's life cycle that they rarely encounter as students: determining clients' objectives and concepts of operation, negotiating and prioritizing requirements, prototyping user interfaces; formulating team project schedules; evaluating COTS products and cloud services; training the system's users; and preparing the clients for evolving the system after its delivery. The paper also summarizes the evolution of the MS program and project course as software processes, clients, applications, components, and tools have evolved, as the program and project course have evolved since their inception in 1993, educating over 2000 students to become considerably more T-shaped and team-oriented than they were on arrival.

Introduction

Between 1973 and 1989, I worked at the Thompson-Ramo-Wooldridge (TRW) company in southern California. My job titles as time went on were Director of Software Research and Technology, Chief Engineer of a major software division, and Chief Scientist of the TRW Space and Defense Systems Group. TRW was a major player in the software field, reaching a high of #2 (behind IBM) in the Datamation list of software companies by business volume. TRW was also the spawning ground for the Waterfall [1] and Spiral [2] software process models.

In the mid-1980's, Simon Ramo created an initiative to expand TRW into non-defense, software-intensive companies. These included companies handling credit data, local telephone exchanges, point-of-sale systems, and industrial process control. One of his coordination mechanisms was a TRW Electronic Systems Technology Advisory Board, of which I was a member. At one of its meetings, he asked each company representative to identify its most limiting factor in growing their business. Most of them said that it was due to a shortage of strong software engineers.

One of the outcomes of the meeting was that Dr. Ramo asked me to work with our major local universities, UCLA and USC, to set up special work-study MS-degree programs for top computer science BS graduates to work at TRW companies three days a week and take courses for MS degrees in software engineering. One of his key priorities was to ensure that the program not just produce super-programmers, but to broaden their courses and work experience. He said, "Our most valuable engineers are T-shaped. They have deep knowledge in their specialty area, but also have working knowledge in their company's application domains, along with knowledge and experience in economics, management sciences, and human factors."

I wasn't sure that UCLA and USC would agree to such a program, but fortunately there were two factors that made them supportive. One was to help them attract top students who were US citizens. The other was that Dr. Ramo pledged to provide a substantial annual contribution to the department chair's discretionary funds.

The program turned out to be a success, as most of its MS graduates stayed with their TRW companies and became some of their leaders. However, most of our new software hires were still I-shaped computer science majors. They had good knowledge of their programming-intensive computer science topics, but relatively little about software requirements, architecture, effort estimation, management planning and reviews, and software qualities such as reliability, availability, maintainability, safety, security, reusability, and performance aspects. Even with TRW training about TRW development and management practices, it was often the better part of a year before they could perform effectively on TRW companies' projects.

Section 2 summarizes the genesis of the USC CompSci/Software Engineering MS Curriculum and its course components, focusing on the key 6-student, real client project course. Section 3 describes the main challenges in organizing and evolving the project course. We encourage MS students to take the project course right away as a foundation stone to their MS degree rather than a capstone at the end, since having the experience of negotiating a project's requirements, developing its architecture, prototyping and evolving its user interface, planning and estimating a project's effort and schedule, identifying and managing a project's risks, and preparing the product for use and evolution, gives them a head start in the other courses, and an appreciation of why each course's content is valuable. Section 4 summarizes how the curriculum and project course have evolved over their current 25-year lifetime.

Developing a CompSci/Software Engineering MS Curriculum and Course Components

In 1989, I became eligible for early retirement at TRW. I could stay at TRW for another 10 years, or do something different. After exploring the prospects with UCLA, USC, and UC-Irvine, and a 3-year hiatus in which I accepted a term position to head the US Defense Advanced Research Projects Agency (DARPA) Information Science and Technology Office, I accepted an offer of an endowed chair in software engineering from USC. So in late 1992, I joined the USC Computer Science Department, and began developing plans for a USC Master of Computer Science degree with specialization in Software Engineering, (MSCS-SE), to begin in the Fall semester of 1993.

Besides taking breadth CS courses in AI, networking, formal methods, and computer graphics, the MSCS-SE core courses included software requirements and architecture, computer hardware-software design, user interface design and development, software management and economics, plus a 2-semester, real-client project course. The project course included students forming 6-person teams to determine a client project's requirements, design, development, verification and validation (V&V) plans, and client transition plans in the Fall semester, and incrementally developing, V&V-ing, and transitioning the resulting system in the Spring semester, including training the users and maintainers, delivering the completed architecture, user manuals, maintenance manuals, and maintenance support tools and regression tests.

The course was quite popular, attracting about 90 students. Finding 15 clients turned out to be difficult, so for the first two years, I served as the client for each of the teams, using example projects from my experience base. In the first year, the students developed versions of a fire department dispatching system. In the second year, the students developed version of a library selective dissemination of information (SDI) system, using borrowers' interest profiles to notify the users about new library acquisitions they might be interested in borrowing.

Fortunately, a couple of months into the second year, I was visited by several of the USC librarians. They were wondering why so many students had come to the library to ask questions about how the library operated and how it stored and processed its information, and they had found out about my course project. They indicated that the various libraries had needs for software systems, and asked if the students could develop software systems that they needed but had no resources to develop for themselves. It was too late to restart the second-year teams, but we agreed to work out processes and plans for our teams to develop such applications in the following year, while they worked with me to identify library clients and library applications that would fit the 2-semester constraints on the projects' size and complexity.

It turned out that many of them had special collections that they would like to provide capabilities for people to virtually browse over the internet, that were good fits for the 2-semester constraints, plus a few special cases where the collections were dynamically evolving, such as the business school's collection of stock market price information, or collections requiring special handling or access restrictions, such as medieval manuscripts. But by the Fall semester of 1995, we had a set of 15 clients and applications that were good fits for the 15 teams to develop and transition in two semesters. Overall, with a few complications, the projects went along very well, and again fortunately, the USC daily newspaper did an extensive article describing the course and projects, which attracted a good many potential USC clients for future years' projects.

Eventually, we started to run out of USC organizations needing software developed for them, but again fortunately, one of the organizations was the USC Neighbors charitable organization. USC's payroll system allows employees to contribute a monthly deduction to support USC Neighbors, which provides funds for organizations helping local minority students to prepare for going to college, stay in school, use the USC campus on weekends to put on and participate in events, learn marketable skills, and the like. The USC Neighbors organization needed a system to track and manage such organizations and activities, and I indicated that our student teams could also provide the organizations with software systems to keep track of their funds, donations, volunteers, events, etc., and to have web sites that people could visit to see what activities had recently been going on or which activities or new initiatives were coming up in the future. This led to a number of projects in which our students could build useful software applications for them, and also led to further applications that the students could build for USC neighborhood small businesses or local government organizations. Currently, we have about 25% each of the projects being done for clients at USC, local charities, local small businesses, and local government organizations.

Course Challenges and Responses

There were also a number of challenges in running the project course that we needed to provide good solutions for, such as forming the 6-person teams, involving distance education students to participate, assigning teams to clients, coming up with fair grading approaches, keeping the projects on track, and evolving the course and projects as new technology came along. These are discussed next.

Forming the 6-person teams. Rather than randomly or arbitrarily assigning students to teams, we made it the students' responsibility to form themselves into teams, making them more accountable for their choices. We had each student post a mini-CV indicating what skills they had, so that if a team needed an experienced manager or database expert, they could search the postings for the best available team member. They had three weeks to form teams; at the end of the second week we had a mating event for unmatched students, with pizza and signs for students with particular skills to stand next to. As the course enrollment was a multiple of 6 only 17% of the time, we would create 5-person or 7-person teams to find everyone a place.

Involving distance-education students. Since they were generally not able to interact with the clients, we assigned them to teams as independent verification and validation (IV&V) teammates. They would review their **team's** artifacts, and submit problem reports. These were tracked via Bugzilla, and either acknowledged and closed with a fix, or identified as non-problems (e.g., a feature desired by the client).

Assigning teams to clients. Often, we would have more clients that student teams, or clients proposing more than one project. During the summer, we have briefings for the clients, explaining their roles in the course, and how we assign student teams to projects. Each team reads the proposed projects, and as part of their registering their team, they indicate which proposed projects are their first through fifth choice. Of course, not everyone gets their first choice, and we have a minimum-regret algorithm that assigns projects to teams that minimizes the sum of the total ranking scores of their choices. We explain this to the clients, and encourage them to write their project proposals to be attractive to the student teams.

Coming up with fair grading approaches. Each team member receives the same grade for their team's performance during each semester, which includes grading of their artifacts by our teaching assistants, who have taken the course, and satisfaction ratings from their clients, which include ratings on questions on a scale of 1 to 5, plus space to include comments. The resulting score will be 50% of their grade total; the other 50% comes from individual homework assignments, plus an essay critique of their team performance and of suggestions for improving the course. The critique accounts for about 20% of their grade for each semester. The critiques are the primary ways we have to determine which team members are contributing the most, as weak contributors generally submit weak critiques.

Keeping the projects on track. Twice during the semester, we hold Architecture Review Board reviews based on the AT&T Architecture Review Board model [3], but taking 80 minutes per team vs. AT&T's 2-3 day reviews. Each team member needs to present the results of their activity. Each team member has a primary and backup role: the roles are Operational Concept and client interface lead; Requirements recorder, including the results of using our Win-Win requirements negotiation tool; Lead system architect; Detailed UML modeler; Life cycle planner; Project manager, including risk management; and the distance-education IV&V performer.

The attendees include the client and any of their end-user representatives; myself and my associate instructor, and our **teaching** assistants. Members of other teams can attend as observers. We also evaluate each presenter's performance as a contribution to their individual grade.

Evolving the course. We rely a great deal on the better course critiques in revising the course material and procedures during each summer, and on general trends in overall software technology, client project trends, and research that is being done by our Ph.D students, who often use the projects as testbeds in testing their hypotheses about how to improve various aspects of software engineering. These trends and their impact on the evolving nature of the course over the last 22 years will be discussed next.

Evolving the Course and MS Program

Section 2 described the evolution of our clients in going from library client to overall-USC clients to USC-neighborhood service organizations, small businesses, and local government organizations. It also summarized the evolution of the project course in terms of its applications, tracking the needs of these organizations. Here we focus on the changes to the project course and the specialty courses as more and more of the clients' needs were satisfied by emerging COTS products, open-source capabilities, and cloud services. In the early days of the project course, the students had to develop their own infrastructure capabilities, such as search engines and web crawlers. Quite soon, versions of these and other services were available as open source capabilities along with database management capabilities and various web services. These enabled the student teams to provide much more capabilities within their 2-semester constraints, and often to finish in 1 semester. Somewhat later, cloud services such as Salesforce.com performed most of the functions needed by small businesses, and many of the functions needed by service organizations, such as keeping track of their events, volunteers, donors, and persons served by charity organizations. Now, most of the projects are able to be completed in one semester. Recently, a new class of clients are becoming frequent: USC entrepreneurs, often students in the business school, who would like a student team to develop an initial version of their new class of service, such as valet parking or specialist collaboration services (e.g., screenwriters).

Another source of project Improvements have come from tools that we have developed and evolved. The COCOMO family of effort estimation models is one example. Another is our sequence of win-win requirements negotiation and prioritization tools, enabling clients and developer teams to negotiate mutually satisfactory or win-win compromises on capabilities and priorities of desired features. These come in handy in dealing with the uncertainties in how much capability can be developed in one or two semesters. Since the COCOMO model produces not only a most likely effort estimation, but also optimistic and pessimistic estimates of 25% more or less effort the project will need, we can determine how much capability can be developed even in pessimistic situations. We can then enable teams to take the prioritized requirements and promise to deliver the pessimistic set of core capabilities at the end of the one or two semesters, but generally will have them available earlier, at an event called the Core Capability Drivethrough, in which the client exercises the core capability and revises their priorities for what can be done for the rest of the semester.

Resulting Benefits

The main benefits are for the students. When they go to job fairs or hiring interviews, they have a portfolio describing the project on which they participated, and can demonstrate their level of T-shaped capability in not only programming, but also capabilities in economics, business case analysis, human factors in prototyping, life cycle maintenance preparation, and domain skills in the domain of their team project. They get better job offers, and the hiring companies subsequently come back looking for more graduates of the MS program. Another main set of benefits come for the clients, who generally receive more capability than they expected, often after one semester rather than two. And for the local charity organizations and local small businesses, the resulting software systems help benefit the local community. Other beneficiaries are our Ph.D. students, who have a critical mass of projects on which they can test their hypotheses. For example, they were able to show that the value-based prioritization of features to inspect and test enabled projects to double the value of the results per hour of inspection or testing. And as instructors trying to keep up with the latest in software technology and incorporate it in our courses, the experiences of applying new technologies to practical projects enable us to keep reasonably up to date with rapidly-evolving software technology and its effects.

References

[1] W.W. Royce, Managing the Development of Large Software Systems: Concepts and Techniques, Proceedings of WESCON, August 1970. Also available in the Proceedings of ICSE 9, Computer Society Press, 1987.

[2] B.W. Boehm, A Spiral Model of Software Development and Enhancement, IEEE Computer, (21) 5 (1988) 61-72.

[3] J.F. Maranzano, S.A. Rozsypal, G.H. Zimmerman, G.W. Warden, P.E. Wirth, D.M. Weiss, Architecture Reviews: Practice and Experience, IEEE Software (22) 2 (2005) 34-43.