

What's Missing in Academia to Prepare Students for the Software Engineering Workplace

Keith Schifferli
MBA, PMP, CSM
MSE Student
Seattle University
Seattle, WA 98122
Schi2706@seattleu.edu

Vijay Singari
MSE Student
Seattle University
Seattle, WA 98122
singariv@seattleu.edu

Dimpy Gill
MSE Student
Seattle University
Seattle, WA 98122
gilld1@seattleu.edu

Abstract

Software Engineering continues to be a top career path for college students entering the workforce as well as existing professionals looking for a career change. However, most are not properly trained in the core skills the industry currently demands.

Practices such as working in teams, inspections, reviews, architecture design and planning are becoming more fundamental in the workplace, yet students are not learning or learning enough of these core practices in their curriculum. This leads to individuals stepping into the workplace, not prepared or setup for success.

Introduction

Money Magazine and Salary.com in 2006 rated software engineering as the best job in America with a 10-year forecasted job growth of 46.07% (CNN Money, 2006) (Appendix A). Some would say that this reinforces software engineering as a profession, similar to how we view professions in sports, medicine, law, and education. Merriam-Webster defines a profession as “a calling requiring specialized knowledge and often long and intensive academic

preparation” (Merriam, 2008). In addition to the standard definition, we can assume that the academic preparation is very well defined, structured, disciplined and standardized to enable the professional to be successful as they enter the workplace. However, when we look at the structure, content and rigor offered in today’s software engineering curriculum, we don’t necessarily see the types of structure we would expect in a profession as we’ve defined it. Specifically, today’s software engineering workplace is not about the individual, it is about the team achieving a common goal through a process of standards and practices. And as the need for businesses to be global continues to grow, fostered by the availability of low-cost offshore resources, the software industry is gravitating towards enabling a collaborative environment through methodologies, tools and disciplined processes. Methodologies like PMI (Project Management Institute), Agile Software Development, SCRUM, and Extreme Programming are gaining popularity for many of today’s project managers and senior executives. Collaborative tools like Microsoft SharePoint, Visual Studio Team Foundation Server, DevPartner Studio and Rational Rose that enable engineering teams to communicate and work seamlessly together continue to be a growing market. Disciplined processes like Earned Value Management (EVM), Team Software Process (TSP), Application Lifecycle Management (ALM) and Capability Maturity Model (CMM) ® are sought after to enable predictability, accurate estimation and on-time delivery with high quality. Yet, with the industry continuing its direction around enabling teams to work towards a common goal, today’s academic environment isn’t training graduates to be prepared for this new world and “...leaves the quality of each individual’s work as a matter of blind luck” (Humphrey, 2005, p. 4). There are many skills, techniques and tools that are required to be successful in the workplace that can and should be made available in the

software engineering curriculum. Experience working with teams that share a common goal and practices such as code inspections and walk-throughs, planning and process along with “designing in the large” at an architectural level are key components that should be learned in the college experience to help the student be prepared for the requirements of the workplace. “It’s important that we blend process and technology, not just technology. They (academia) need to ingrain the principles like code inspections.” (Janardanan, 2008)

As a real world example of the impact, we looked at two separate sources of live information. The first was a three person project team working on an masters-level Capstone project at Seattle University. Many observations were made throughout the team’s lifecycle that will be referenced as real life experiences. And while the team is small and may not represent an average team and project size that would be experienced in a working environment, it serves the purpose of providing specific examples relative to this discussion. The second source was a series of interviews with ten TSP Coaches working at Microsoft in the company’s various Information Technology organizations. The coaches were able to offer direct observations of the teams they oversaw; specifically where there were opportunities for improvement that should have to be taught in college.

Working in Teams

Today, an aspiring software engineer in college is expected to predominantly work as an individual. Whether it be programming, developing project plans or writing a paper, the focus is on the individual’s work. It is rare to see true team collaboration in this environment, which is contradictory to what is expected in the industry. One immediate benefit of working in a

team environment is the experience gained in the use of collaborative tools. As individuals we typically do not leverage these tools because they seem like overhead, when solely focused on an individual deliverable. However, when communication, sharing and collaboration become a driver for success, these tools enable us. Simply having experience with these tools can make the transition from an individually focused to a collaborative and partnering environment, a less painful experience. Tools like software source control and configuration control can be easily used by students working in teams and something they will immediately begin using in their job when they enter the workforce.

The value proposition of practicing teamwork in the academic environment exists not only with respect to learning *how* to use the tools; but also offers development of interpersonal skills. As individuals experience working with others towards a common goal, practices like negotiation, open and honest discussion, giving and receiving feedback, transparency in purpose and managing others without any level of authority become learned and well honed skills.

Experience has shown that most people that transition from the individual to the team environment find social skills – not the technical skills - to be the most difficult to learn and adopt. By exposing students to these specific team situations, we not only gain the above mentioned experiences and skills, it also helps to enable the team-level disciplined processes that are becoming more common in the workplace. Additionally, when we look at the **Forming – Storming – Norming – Performing** model for group development proposed by Bruce Tuckman in 1965, we see that going from Storming to Norming (a team that is dysfunctional to a team that is functional) offers the greatest challenges for teams. In order to be successful, individuals need the skills to work within a team environment, on their individual responsibilities as well as

their ability to work with other individuals that have different styles and communication patterns. Experience is a key enabler for individuals working collectively as a team to move out of a storming to a norming mode in the model and in real practice. “The principles need to be in their DNA. Team concepts as opposed to individual concepts.” (Smith, 2008).

The team working on the Seattle University Capstone project spent a lot of time in the storming phase. This was due to several factors that can be mainly attributed to a lack of experience working on a project team where there are individual and shared responsibilities relative to the project. As an example, two of the team members had very limited knowledge or experience in working with software source control. In the project this led to an initial disruption and delay due to out-of-date code in Microsoft SourceSafe and an ongoing issue where key components were checked out to a single individual for extended periods of time, which blocked other team members. It was also observed that very little of the code was commented (documented) in a manner that would be expected from a team coding collectively on the project. Much of this can be attributed to their primary experience of working in isolation on their code. The team environment required them to share code so others could work on it, which was not an easy rhythm to step into.

Inspections and Walk-throughs

A typical student’s coding experience involves a small amount of minor fixes; they can get their program up and running, which doesn’t necessarily equal high-quality but leaves the individual believing that because it works, the quality is enough to meet the standards (a good grade in the class). When personal reviews are done in this type of scenario they are

predominantly unstructured and fail to yield a high success rate in finding and correcting defects because they lack appropriate planning and discipline. In addition, we are trained to rely on tools like the compiler to catch our mistakes, but these are not 100% successful in most cases (Humphrey, 2005). Today's more mainstream methodologies promote disciplined practices that enforce detailed reviews of an individual's work. Two of the more common techniques are code inspections and peer reviews. Code inspections was introduced in 1976 by Mike Fagan and is an example of very structured team reviews. A walk-through is less formal than an inspection but still brings the team together for a presentation of the work product for discussion and critique. With inspections and walk-throughs, an individual's work is now under scrutiny for defects, using a well defined and disciplined process. If we think about the context of the typical college class, individuals are in competition with each other for grades (assuming a grading distribution is used) and inspections or reviews are only done by the professor of the class. The student now enters the workforce and experiences code reviews and inspections by his/her peers and may not have made the transition from viewing themselves as competitive peers to a team that is working towards a common goal. This makes the disciplined practice difficult to learn or adjust too successfully and can cause difficulties in teams. Students need a good foundation for large-scale, multi-person and globally distributed project teams.

The Seattle University team experienced this as they were reviewing each individual's work product. The reviews were not as productive as they could have been, since the practice of giving and receiving constructive feedback without fear of repercussion was not an experience most of them had. Many defects were later discovered in work products because the issues

were not brought up during the formal reviews, which lead to rework and “design as you go” methods.

Planning and Process

A key element to successful execution is having a well thought out and detailed plan. Students are not taught how to approach planning with respect to the work they are required to deliver. For the most part, they are taught technical skills which according to a Standish report (Standish, 1999), isn't a key component to delivering on a successful software development project. Take a simple programming assignment given in any standard class. The student is taught the programming language, proper coding structure and the conceptual models for programming. However, they are not taught or required to formulate a plan for how they are going to program. Not only are students not taught to focus on planning, but they are made to believe that planning is unproductive. It's engrained on us at an early age to deliver results and writing code feels productive where planning doesn't; coding is visible progress.

With a disciplined approach, the work goes through a conceptual, logical and physical design based on the requirements. The components are then defined into packages and the subsequent tasks to deliver on those components can be broken down into specific items. Incidentally, this approach can be seen in Agile methods like SCRUM and the Team Software Process (TSP) developed at Carnegie Mellon. Detailed planning like this enables the software engineer to truly understand the work effort required for the deliverable. It also allows a more

refined estimate that can later be compared to actual effort at a detailed level and used to estimate future work that contains similar tasks.

This type of disciplined planning enables success as the person enters the workplace, because they have learned how to think through the problem using a reliable and defined process.

When they are asked to provide estimates, they can be more accurate and then deliver successfully, thereby being successful in their job. They are also made successful in discussions that involve executive or senior management, where they are brought in to explain the estimates or articulate why expectations are different than what management believes is realistic. This is based on having experienced (in the classroom) fundamentals like the traditional triple constraints in the Project Management Triangle (see Appendix B) and how to validate or negotiate unrealistic commitments (Mohan, 2008).

Architecture & Design

“The lack of a precise design is the source of many implementation errors” (Humphrey, 2005, p. 226). Not only does planning and process enable better estimates, it also drives the engineer to think in terms of “programming in the large”, as opposed to “programming in the small”. When “programming in the small”, the developer is focused on constructing individual modules as opposed to “programming in the large” where we think about the system and how the modules interact. In academia, it is possible to achieve a high degree of success with little to no formal design because the programs are small enough to be well understood by the sole individual that is responsible for the work. In the workplace where the projects are much larger and more complex, the ability for an individual to completely and clearly understand a

program's design is virtually impossible. Disciplined planning and process enables the ability to think through the entire problem through the use of design techniques and consideration towards the architecture of the end result, "Solution development versus programming" (Brisse, 2008). Another way to look at it is rather than immediately starting the work and "figure it out as you go", you invest the time to think through the problem and architect or design the solution to meet the goal. This leads to true design before coding as opposed to the more common "design as you code". This type of experience is not usually gained in college because the focus is predominantly on the individual and their module or work package. Yet, it is expected in the workplace and usually as a team effort.

In a 1999 Chaos Report published by the Standish Group (see Appendix C), it was demonstrated that the likelihood of success (on time, within budget, requirements met) increased as the project and team size decreased. If we take this to the level of an individual working on a personal project in college, their likelihood of success is consistently very high and that is the experience they gain in their tenure as a student. That same student enters the workplace, working on a larger project with additional people involved, success no longer comes easy and they are not prepared for that experience.

Conclusions

Software Engineering continues to grow as a career destination of choice with little sign of slowing down in the immediate future. Today, not much is offered in academia to provide standards that are consistently followed across the profession. Software developers are left to figure out their own methods and standards without the guidance and support that other

professionals receive (e.g. Sports, Medicine, and Law). Key skills like the value of planning and design can be taught and improved with practice, something that should be learned in the proper environment as opposed to on the job. The individual needs to acquire disciplined skills that can scale from small college level projects (a few thousand lines of code) to large, corporate sized project (1 million lines of code). Examples of this are code inspections and walk-throughs that determine root causes of defects as opposed to standard testing measures that address the symptoms the defect causes. As a corollary, disciplined approaches tends to lead to higher quality products which would further enable the individual to provide value to the company by which they are employed.

Finally, the value of learning how to work within a team environment is critical to success. The paradigm shift an individual has to make when going from an individual-centric to team centric environment can be a huge leap for most; one that should not be required as “on the job” experience.

Failure to instill discipline into this profession based on proven methods will lead to continued problems in the industry with respect to the ability to deliver successful projects of high quality. "Those who cannot remember the past, are condemned to repeat it," (Santayana, 1905). The starting point is a focus on a team-centered culture in academia.

References

Babu Chandra Mohan, Priya. Personal interview. 28 February 2008

Balmuri, Sharath. Telephone interview. 6 February 2008

Bhosale, Sudhir. Telephone interview. 7 February 2008

Brise, Frank. Telephone interview. 29 February 2008

Chadalavada, Anantharam. Telephone interview. 13 February 2008

" Forming-storming-norming-performing." Wikipedia.org.

<http://en.wikipedia.org/wiki/Forming-storming-norming-performing>.

Humphrey, Watts S. PSP A Self-Improvement Process for Software Engineers.

Upper Saddle River, NJ: Addison Wesley, 2005.

Jain, Mukesh. Telephone interview. 6 March 2008

Janardanan, Radhika. Telephone interview. 7 February 2008

"Merriam-Webster Online." 2009. Merriam-Webster.com.

<http://www.merriam-webster.com>.

" MONEY Magazine and Salary.com rate careers on salary and job prospects." 2006. CNN

Money.com.

<http://money.cnn.com/magazines/moneymag/bestjobs/2006/top50/index.html>.

Santayana, George. The Life of Reason. Old LandMark Publishing; New Ed edition, 2005.

Smith, Jeff. Personal interview. 15 February 2008

Standish Group, Chaos: A Recipe for Success, The Standish Group Int'l, 1999;

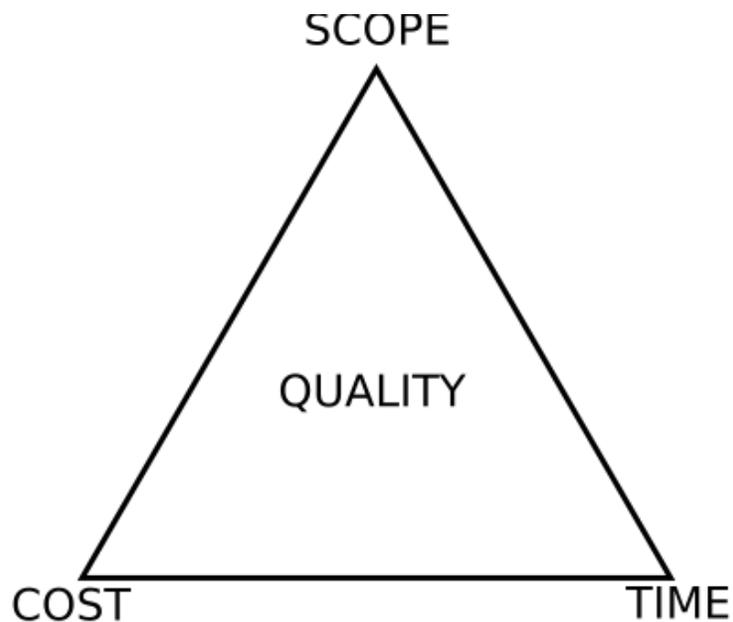
www.standishgroup.com/sample_research/PDFpages/chaos1999.pdf.

Warns, John. Personal interview. 6 February 2008

Appendix A

BEST JOBS IN AMERICA		Money salary.com™	
MONEY Magazine and Salary.com rate careers on salary and job prospects.			
The Top 50		More jobs: Stats on 166 titles	How MONEY picked the best jobs
Rank	Career (click for CNNMoney.com snapshot)	Job growth (10-yr forecast)	Average pay (salary and bonus)
1	Software engineer	46.07%	\$80,427
2	College professor	31.39%	\$81,491
3	Financial advisor	25.92%	\$122,462
4	Human resources manager	23.47%	\$73,731
5	Physician assistant	49.65%	\$75,117
6	Market research analyst	20.19%	\$82,317
7	Computer/IT analyst	36.10%	\$83,427
8	Real estate appraiser	22.78%	\$66,216
9	Pharmacist	24.57%	\$91,998
10	Psychologist	19.14%	\$66,359

Appendix B



Appendix C

Project Duration, Team Size Affect Project Success

Project Size	People	Time (mos.)	Success Rate
Less than \$750K	6	6	55%
\$750K to \$1.5M	12	9	33%
\$1.5M to \$3M	25	12	25%
\$3M to \$6M	40	18	15%
\$6M to \$10M	+250	+24	8%
Over \$10M	+500	+36	0%

The smaller the team and shorter the duration of the project, the greater the likelihood of success. Obviously, this does not suggest that compressing the schedule and reducing the resources of a large project will make it successful. Nor should it be construed that large projects with large teams cannot be successful. The Standish Group believes any project can be successful if all the key criteria are met.