Pulling It All Together
Preparing Software Engineering Students for the Real World

Lee Vallone,
Monmouth University
West Long Branch, New Jersey

ABSTRACT

"The major problems of our work are not so much technological as sociological in nature." So reads the self-proscribed thesis for DeMarco & Lister's landmark book, Peopleware. In most development organizations managers spend most of their time solving technical problems, instead of addressing the political & interpersonal issues that are the real culprits for the high failure rate of software projects. Similarly, we teach our students the virtues of use-cases and spiral development models, frequently ignoring the true keystones of student success in the project world: teamwork, collaboration and the processes that tie everything together.

There are really two objectives for this paper. One is to describe a method for teaching process, quality and measurement in a way that is engaging and enables students to really internalize the material. The second objective is to describe an approach that helps students understand (and experience) the role and importance of sociological issues and how to address them in a way that substantially increases the probability of project and personal success. It is based on the premise that software engineering is so much more than the technical disciplines.

Introduction

"The major problems of our work are not so much technological as sociological in nature." So reads the self-proscribed thesis for DeMarco & Lister's landmark book, Peopleware (DeMarco and Lister, 1999). In most development organizations managers spend most of their time solving technical problems, instead of addressing the political and interpersonal issues that are the real culprits for the high failure rate of software projects. Similarly, we teach our students the virtues of use-cases and spiral development models, frequently ignoring the true keystones of success in the project world: teamwork, collaboration and the processes that tie everything together.

In software development organizations, process is a dirty word, the dreaded 'P' word, and most software engineers view it as the stereotypical Dilbert style waste of time. To get compliance and participation, true believers and Software Quality Assurance teams (the process police), frequently resort to the threat of ISO and/or TL 9000 non-compliances. In most cases, compliance obtained in this way amounts to lip service. The net result is that the processes provide little value and the poor results only serve to reinforce the impression that this "stuff" is useless and detracts from the real objective, writing code. From my background on various industry projects, I believe there are really only two cures for this downward spiral: personally experiencing failure, and personally experiencing success.

The course described in this paper is set up to do exactly that: put students in a position where they experience the value of good project processes and also experience the failures that occur as they get into crunches and resort to seat of pants or hero methods. But, in the process of achieving this, the students also learn the importance of teamwork, leadership and trust. They learn how teamicidal behaviors and individual personalities can destroy technically outstanding teams. And especially, they learn to view mistakes and failures as opportunities to improve for the next time.

The Challenge

As educators, our challenge is to teach software engineering students the skills necessary to survive and thrive in the real world of project developments. So what are those skills? If the average graduate level curriculum is any indication, they must be: Use Cases and Object Oriented Design & Analysis, Test Methods & Procedures, Agile Software Engineering, Requirements Engineering and Software Project Management. Right? Wrong! I contend that while necessary, these skills play only a relatively minor role in future success.

The real skills software engineers need to succeed include: teamwork, communication skills such as listening & supporting, conflict resolution, leadership, risk management, presentation, political skills, and finally, knowing how all the technical and interpersonal pieces fit together.

Most software engineers learn these skills the hard way, by trial and error on the job. Some never learn. Consider the typical engineering career progression where a developer does an outstanding job designing and implementing software and is thus promoted into management, a position that relies almost entirely on the "soft" skills. While one might argue that the high failure rate of software projects is due to aggressive schedules and unrealistic customer expectations, it is certainly conceivable that the Peter Principle[1] is a significant component of many software project failures.

So, how can we teach these skills in the classroom? Lectures may help expose students to the key interpersonal issues, but by far, the most effective learning method is for the students to experience a project development from start to finish. Even better is to iterate and apply the lessons from previous attempts.

Approach

Monmouth University's SE652 attempts to do exactly this. The course is billed as covering software quality, process and measurement. It utilizes pieces from several widely accepted approaches to fulfill these topics and also accomplish the real mission of helping students understand and experience the role and importance of sociological and process issues. The basic approach is to create small teams of students, give them an aggressive project objective and allow them to fail! Of course, the key is to ensure they have enough qualitative and quantitative

[1] The **Peter Principle** is a theory originated by Dr. Laurence J. Peter which states that employees within a hierarchical organization advance to their highest level of competence, are then promoted to a level where they are incompetent, and then stay in that position.
Wikipedia Free On-line Encyclopedia: http://en.wikipedia.org/wiki/Peter_principle.

data from the failures to actually understand what went wrong and how to avoid similar problems in the future.

The project component and overall structure of the course follows the *Introduction to the Team Software Process: TSPi* (Humphrey, 2000). The TSPi outlines several small projects and provides a wealth of supporting materials including process & role descriptions, checklists, tools and instructor guidelines. On the downside, the process approach is heavyweight and requests what is arguably an excessive amount of data collection that obscures the considerations and value of a solid measurement program.

Early offerings of the course started with the TSPi measurement program but the course has since diverged from this approach and now follows the measurement strategy outlined in *Practical Software Measurement* (McGarry, et.al. 2002). Practical Software Measurement (PSM) is an excellent framework for creating measurement and data collection plans and implementations. In contrast to the one-size fits all approach of the TSPi, the PSM guides practitioners to be selective about the data collected / measures tracked and to tailor the measurement plan to the specific project risks and objectives.

Interpersonal experiences and lessons are integral to the course and DeMarco & Lister's *Peopleware* is a natural companion to the more technical texts. The authors cover topics such as trust, teams, quality, hiring and much more in an entertaining style that really drives home the considerations and achievements necessary to deliver successful projects.

Teams

The first task in the course is to create the teams. Similar to industry projects, the students do everything with their respective teammates. Teams are a key element in the success and failure of any project, and we want to drive this lesson home. The students ultimately need to learn each other's strengths, weaknesses and idiosyncrasies, hopefully learning to leverage the diversity of the team so as to create a whole that is much larger than the sum of its parts. The teams must be accountable, responsible and empowered. Motivation and shared accountability is achieved through the typical university standby, grades. The biggest component of each student's grade is determined by team performance on the project. Each team is solely responsible for its commitments and performance. The teams are given guidelines on the grading criteria (quality, schedule, functionality, quality records), but are then empowered to choose their own path. The team decides on how much functionality to commit to, how to organize, what to measure and how to divide up the work.

The course tries to kick-start team formation and gelling via a Survival Simulation (Survival Simulations, 1987, et.al.). Survival exercises are lots of fun and help students understand team dynamics and how specific individual behaviors can help or hinder team performance. Teamwork is a recurring theme in the course, and Survival helps introduce several key topics such as: leadership, leveraging diversity, dealing with conflict and communication styles/behaviors.

Current offerings of the course have allowed the students to pick their teammates, with the only constraint being the team size (5-6 students). The result of this approach has been mixed. In

many cases, choosing their own teammates has sped up the teaming process since many team members had previously worked together. On the other hand, students tend to be unfamiliar with how to create effective teams, frequently using familiarity as the #1 criteria, rather than skills or diversity. The net affect has been some dysfunctional teams and in a couple of cases, teams without key skill sets (e.g. someone with leadership skills).

Several of the weaker teams have been easy to spot from as early as the second or third class. But, there have been some interesting surprises too. One team that appeared to be highly dysfunctional turned out to be the top performer as measured by delivered functionality, quality level and schedule performance. This team was especially noteworthy because of the process they went through to gel, starting the course with major interpersonal conflicts (e.g. shouting matches) and frequent in-fighting. By the time the course ended, this team was thriving on those same differences and delivering results much greater than those same individuals could accomplish working independently. Another team in the same class offering was notable for being on the other end of the spectrum. This team initially appeared to be one of the stronger teams with excellent individual skill sets that seemed more than adequate to the task at hand. Unfortunately, this team ended up totally "blowing it", missing key milestones and delivering sub-par quality. The difference? They never gelled. There were probably several reasons the team never gelled, but one in particular was their inability to subvert their egos and deal with conflict. When disagreements arose, they were either swept under the rug or they would "agree to disagree" (e.g. you go your way, I'll go mine). As stated, the results were disastrous and they turned out to be less of a team at the conclusion of the course than they were at the beginning!

The Project

The team objectives are to deliver a specific tool, meeting a vaguely defined need, to a defined customer (the professor), on time, with demonstrated quality and ensuring that a full set of quality records and performance data is collected and utilized along the way.

A course objective is to mimic, as closely as possible, an end-to-end real world project development and many attributes of real work projects occur naturally in the classroom. Aggressive schedules and commitments occur without prompting as students consistently underestimate the work. Familiar surprises occur as teams lose members to business trips and other unexpected life events. Risk and reward trade-offs are encouraged through the grading system as greater commitments provide the opportunity for higher grades. The course even goes so far as to introduce performance reviews as a tool for empowering the team leaders (though it also introduces peer reviews as a tool for gaining feedback on all of the team members). Roll this all together and one other staple of software development is in ample supply, conflict – both intra and inter-team!

Process Flow

The course follows the standard process steps outlined in the TSPi: Concept/commitment, requirements, design, implementation, integration, verification and post-mortem. Over a twelve week summer semester, the course squeezes in two iterations so students have the opportunity to learn and improve. One significant variation from the TSPi is the introduction of cross team testing (i.e. the role of System Test is performed by another team), mimicking the concept of

independent test organizations. While this deviation complicates the logistics, the dynamics introduced are well worth the variation.

Measurement

As mentioned previously, the first offerings of the course used the measurement plan and constructs from the TSPi. This approach proved overwhelming, resulting in unnecessary data being captured with insufficient focus on the characteristics unique to the course project. For example, in an effort to teach schedule tracking, the TSPi asks students to collect and compute Earned Values. The problem is, the project schedule is fixed and the computed Earned Value vs. Planned Value is rather uninteresting. In addition, the TSPi does not tie the measurement plan to the project risks, nor does it structure the plan to support early indication of impending problems. This approach to measurement runs counter to the objective of applying process and metrics specific to the unique characteristics of each project.

The Practical Software Measurement (PSM) based plan is simpler and requires less effort, but it does require greater understanding of the project characteristics. The basic approach: start by identifying the project risks, identify the key measures and data to be captured, define the early warning thresholds, use the data in the post-mortem to assess performance and update the measurement plan for the second cycle. The analysis required to pick appropriate metrics has proven useful in furthering student understanding of the data as the project unfolds and identifying problems.

Unfortunately, this major deviation from the TSPi text introduces confusion since the students can no longer blindly go to the text for guidance on next steps or use the TSPi Excel spreadsheet tool to compute the measures. This is especially true for the first project cycle where the students don't have personal experience to look back on in choosing metrics. On the other hand, the uncertainty introduced is more typical of industry projects.

Presentations

Team presentations are an essential element of the class for many reasons. One key benefit is that public presentation of work creates incentive to produce a higher quality product (e.g. a more thorough analysis of the data and determination of failure root causes). Other benefits include opportunities to polish individual student presentation skills and shared learning across the entire class. During the semester, teams present on a variety of topics including the team commitments and plan, the cycle results and finally, the lessons learned. While the students are encouraged to make some qualitative assessments, it is required that they also leverage quantitative data for much of their analysis (e.g. rework as a function of defect ratios). Experience has shown that initially the non-presenting students are hesitant to speak up and comment on the work of others, but once they loosen up and contribute, the resulting dialogues are an excellent forum that significantly increases the value of the presentation.

Student Post-mortems

The project post-mortem is the time for the students to reflect on the team performance for the cycle and evaluate opportunities to improve. The written post-mortem follows the TSPi

guidelines and goes into great detail about the roles and experience of the individual students along with the overall performance of the team. Similar to the thrust of the TSPi, this document focuses on the process experience. As a companion to the written post-mortem, the teams are also required to do several presentations on each cycle's results with the primary direction that they think about specific notable aspects of the experience and then show data that supports the experience.

For example, if there was a large amount of rework and firefighting during the System Test phase of the cycle, then suitable data might include defect densities, phase/process yields and design/code review results. The resulting discussion would then focus on how the team could have established measurements and early warning indicators to alert them of possible quality problems earlier in the cycle along with potential mitigation strategies.

These presentations have turned out to be excellent discussion and learning opportunities for the entire class, both presenters and audience. They become an opportunity to use real experiences to link performance (and failures) to data; an opportunity to highlight the value of good measurement constructs and even to look beyond the data to other factors for indicators of pending project issues.

Ultimately, the students capture the improvement opportunities and for cycle 1, update the team process documents (e.g. the measurement plan) to help improve the team performance in cycle 2.

Process Assessments

The second offering of the course added an ISO style assessment at the end of the second cycle. The original idea was to support grading the team projects. The real value turned out to be so much more. In addition to experiencing the rigors and expectations of an ISO audit or SCAMPI assessment, the process of following the threads from start to finish (e.g. from an identified requirement through the implementation to the ultimate test results) along with the questions and supporting quality records really drove home many of the course concepts. In fact, the students seem to deem this assessment so valuable, that one of the top course improvement requests was to perform an assessment at the end of the first cycle too.

Class Time

In general, each class has three different segments: project related discussions, lecture topic(s), and team presentations or other project activities (e.g. requirements inspection).

Project discussions are intended to help guide the teams through the project development process. These discussions cover topics such as requirements development, test strategies, measurement, upcoming deliverables, etc. The TSPi scripts are excellent in this regard, but even so, class review of the materials has proven necessary.

Lecture topics are selected to either emphasize important concepts that are germane to the current project phase or topics that may be of use later in the students' careers (e.g. Death March projects, ISO, CMMI). Examples of opportunistic project topics include:

- Empowerment and commitment,
- Risk management,
- Estimation,
- Configuration management and baselining,
- Quality Records,
- And Inspections.

As the course has matured, in-class exercises have become more frequent events. These hands on exercises are partly necessary to keep the students engaged given the class scheduling (single 3.5 hour weekly lecture). But, it has also become clear that experiencing a facilitated process is more instructive than trying to learn from the texts. For example, the TSPi has an excellent description of an effective code inspection process, but observation of various team inspections left one with the impression that the teams didn't quite get it. These inspections tended to be poorly run, inadequately documented and ultimately not very effective. To correct this problem, the class now includes an in-class facilitated code inspection for a known product (i.e. known defects to enable measuring review effectiveness). This exercise allows the students to experience all aspects of a well-run inspection from planning, to constructive dialogue to final documentation and measurement of the results.

Examples of other in-class exercises include: requirements inspections, negotiation games (Thomsett, 1996), NCSL counting, Survival and Prisoner's Dilemma.

Conclusion

While only two offerings old, the student enthusiasm for this course has been tremendous and overwhelmingly positive. I can think of no better way to portray this then in the students' own words from anonymous, post-course feedback. The following is just a sample of these insights but indicate the course is achieving its true objective, the understanding that software engineering is much more than just the technical stuff.
[Samples of verbatim feedback from 2004 course offering – Monmouth University, SE 652]

"Lots of time at work is spent mismanaged and fighting fires. It is a shame that the people providing us with fires and slipped schedules can not have the experiences we have had."

"I had a good time just learning how each of the personalities in our group worked together. Being able to depend on one another and help each other out is very important, and the division of roles was very interesting once we knew what we were doing enough to simply rely on each other."

[In response to 'was the class useful?'] "This class might be the first time I say yes and mean it, I actually learned useful leadership skills and developed an understanding that process, even though it can be a pain in the --- is useful and helpful."

"Most Software Engineering students may know how to produce a SRS or SDD, but [it is] very rare that students know how each phase affects the other phases and the quality of the product as a whole."

"…I feel very confident in developing a much better quality software product than I used to as a result of taking [this] class. I think the knowledge gained in the class will be very useful in later courses and also in my career. This class had a direct effect on my career. I doubt I could say the same thing about any of the other courses I have taken so far."

While the student feedback is encouraging and certainly an important metric on a course's value, a key question remains as to whether the students have internalized the materials and can apply the learnings to future projects and situations. One insight we have into the answer is the course final exam. For the final, the students are given a description of a project and asked to describe the considerations and actions required to lead the project to a successful result. They are asked to consider hiring, development strategy, risk identification & management, metrics, processes, politics and dealing with surprises. The results have been less than stellar. The students tended to be good at regurgitating book and lecture materials, but less capable at applying the underlying principles to the unique characteristics of a project. While it is unknown how much to expect students with little industry experience to take away from a one semester course covering such a wide breadth of topics, I believe there is much more that can be done to improve the course's effectiveness.

But, even if the course never improves on this objective, it has had another unintentional benefit. Observing the team dynamics and the frequent ah-ha experiences of the students, coupled with glowing feedback for a course that is acknowledged to be one of the heavier workloads in the Software Engineering curriculum, has made this a truly incredible experience for me as the instructor and provides motivation to continue teaching topics known to put most software engineers to sleep!

References

Demarco, Tom and Lister, Tim, *Peopleware:ProductiveProjects and Teams,* New York: Dorset House Publishing, 1999.

Desert, Jungle & Sea Survival Simulations, Human Synergistics, (copyrights 1987, 1992, 1994).

Humphrey, Watts, *Introduction to the Team Software Process.* Addison-Wesley, 2000.

McGarry, John; Card, David; Jones, Cheryl; Layman, Beth; Clark, Elizabeth; Dean, Joseph; and Hall, Fred; *Practical Software Measurement.* Addison-Wesley, 2002.

Thomsett, Rob, *Double Dummy Spit and Other Estimating Games.* American Programmer, June, 1996.

Author Bio

In addition to teaching at Monmouth University, the author, Lee Vallone, is currently employed by Lucent Bell Laboratories. Lee has been leading development teams for over 15 years with teams ranging in size up to 70 developers. He has successfully led these teams to an 80+% project success rate as measured by delivering projects on time, on budget and with the required functionality. Most importantly, these projects have also achieved market success as evidenced by customer sales, profit margins and quality criteria such as customer "willingness to repurchase" scores. Lee is currently leading a team tasked with delivering Lucent Technologies next generation optical systems.