Software Engineering Education at Carnegie Mellon University: One University; Programs Taught in Two Places

Ray Bareiss and Mel Rosso-Llopart
Institute for Software Research, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213

ABSTRACT

Teaching Software Engineering to professional master's students is a challenging endeavor, and arguably for the past 20 years, Carnegie Mellon University has been quite successful. Although CMU teaches Software Engineering at sites world-wide and uses different pedagogies, the goal of the curriculum -- to produce world-class software engineers -- remains constant. This paper will discuss two of the most mature versions of Carnegie Mellon's Software Engineering program -- the main campus program and its "daughter program" at the Silicon Valley Campus. We discuss the programs with respect to the dimensions of curriculum, how students work and learn, how faculty teach, curricular materials, and how students are assessed to provide insight into how Carnegie Mellon continues to keep its programs fresh, to adapt them to local needs, and to meet its goal of excellence after 20 years.

Keywords: Software Engineering Education, Learning by Doing, Coached Learning, Mentors, Project-Based Curriculum.

1. INTRODUCTION

Learning is experience. Everything else is just information.
— Albert Einstein

In 1989, Carnegie Mellon University, in conjunction with the Software Engineering Institute (SEI), embarked on the creation of a professional master's degree program in Software Engineering. The educational vision was to base a professional program on three components: core knowledge, practical demonstration, and broad study through electives. Although the curriculum has been adapted to multiple campuses around the world (Silicon Valley, Korea, India, and most recently Portugal), the program goals continue to remain the same: to provide the best Software Engineering education in the world and to produce world class Software Engineers. Twenty years later and after hundreds of graduates, the program has become a standard by which software engineering education programs are evaluated world-wide.

This paper will present the two senior programs in the Carnegie Mellon Software Engineering masters' degree offerings and explore how these programs have evolved somewhat differently to achieve these goals in two different environments.

2. THE MAIN CAMPUS PROGRAM

The original (often referred to "main campus") program continues today as a major benchmark for Software Engineering education. The program currently offers two primary Software Engineering degrees, the Master of Software Engineering (MSE), a 16 month full-time degree, and the Master of Science in Information Technology with an emphasis in Software Engineering (MSIT-SE) degree, a 12 month full-time degree. Both degrees are also offered part-time.

As noted, the curriculum has three components: core concepts, broad elective knowledge, and project demonstration. Five courses convey the core concepts: Models of Software Systems, Methods: Deciding What to Design, Managing Software Development, Analysis of Software Artifacts, and Architecture for Software Systems [1].

What makes the program unique is that the core course concepts and the elective courses are tightly intertwined with ongoing real-world Studio and Practicum projects to allow the students to acquire new skills and apply them on their projects under the guidance of a mentor/coach immediately. Thus, the program is built on the foundation of student teams actually producing software; MSE teams start their Studio projects on the first day and finish their projects 16 months later. (MSIT teams do shorter Practicum projects. The roles of both projects in the curriculum are similar; thus this paper will focus primarily on the MSE Studio.) This approach, of incorporating the project use of knowledge as it is being acquired from the curriculum, can at sometimes appear unstructured, but it enables the students to see how real world projects are affected by the experience and learning of team members. The MSE Studio represents 40% of the program's curriculum.

The main campus pedagogy combines traditional classroom learning with *in vitro* class projects, case-studies, and simulations to emphasize ideas and with *in vivo* use of the material on Studio projects with mentors and a customer to deliver a product. The program has continued to evolve with the MSE now using the ideas of a Proposal Based Studio [2], Academic Project Ranking [3] and game playing to learn concepts [4] as examples of recent changes. Many of the courses are captured for delivery at a distance. (Details of the distance curriculum are beyond the scope of this paper.)

The curriculum is grounded in the Capability Maturity Model (CMM) which was developed at the SEI [5]. Because of the CMM influence, a primary theme of the program is that "data is king" and "If you cannot measure it, you cannot improve it (Lord Kelvin)." This is also a basis for many of the improvement ideas we teach in software process modeling and management [6]. Particular technologies are used to provide context and emphasize ideas, but they are not core to the program, given the pace of change in the industry. Throughout their project work, faculty expect the students *to show* that they are improving, not just *to feel* they are doing better or to *like* what they are doing. An often-repeated quote which captures the philosophy of our teaching is "If you fail, but you know why, you have succeeded; if you succeed, but you don't know why, you failed."

Our teaching materials define the solution spaces in terms of *people, processes, and technologies*, which together set the context in which a software developer creates a solution for the problems at hand. Most developers have a good handle on technology; they have technical skills and generally have confidence that they can learn new technologies quickly if needed to support a software effort. However, they are often less confident about the other two, people and process, so consideration of these permeates the curriculum. We teach the students how to examine, evaluate, and define the right processes, and to implement the right level of process and the right measures to insure the processes are working

correctly for the environment. We also teach the student to use the differences in people to create strength in their teams. Team exercises, instruments such as the Myers-Briggs and team role exchange help teams to understand and appreciate individual strengths and weaknesses.

How Students Work and Learn

Students typically attend class twice a week for each class -- about 3 hours total. Most core courses are designed for 12 hours of work per week so this leaves 9 hours of outside work the students are expected to perform to understand and apply the knowledge. Usually the students are expected to prepare for class by reading material assigned on a course website and answering reading questions to insure that they are prepared before class. In addition to faculty lectures, class sessions are often student-driven, featuring question-and-answer discussions and presentations. Students calibrate their own knowledge by the questions (and answers) of others.

As previously noted, assignments involve both class work and outside projects. These can be individual or team based. All courses provide *in vitro* project work to insure practical understanding of the material presented as students apply the material and check their initial understanding in an academic or research type project.

After gaining *in vitro* mastery, the students are expected to apply the material *in vivo* on their Studio projects under the guidance of a faculty mentor/coach. Mentors are deeply aware of the curriculum and guide the students in trying out learned concepts when project opportunities arise. In particular, the Studio Project [2] is the "gateway project" of the MSE program (called this because a student cannot graduate from the program without its successful completion). The Studio is a significant software effort scoped for five or six students to complete (while practicing key concepts) over four semesters. The students are assigned two mentors and commit to 12 hours per week for three semesters and 48 hours per week during the summer session. This represents a significant portion of the MSE degree (again, 40% of the student's time).

Finally, students are asked to reflect on both the *in vitro* classroom and the *in vivo* Studio experiences to reinforce and generalize their knowledge. This also brings the results of application of knowledge back into the classroom to provide feedback to the faculty on its relevance and usefulness on a real project.

A significant portion of the student work is written, requiring students to have very strong writing skills to do well in the program. Whether writing or speaking, students are expected to be able to defend their answers -- even when reusing information or examples presented by the faculty.

How Faculty Teach

The primary learning context is the "traditional" classroom. Faculty are expected to present key concepts and interesting examples in the classroom at regular intervals throughout the semester. Class sessions regularly mix instructor lectures, guest subject matter experts, and classroom discussions.

The use of lectures forms a path through the material, but one key notion for students is that we are showing *a way* through the concepts, but not *the only way*. Students who develop new ideas and new ways of applying them are viewed as exceptional, and many times their answers become exemplars for other students to review. Instructors put a premium on reflection, for example, what students might do differently the next time they are faced with a similar problem. What were the critical ideas or inputs that the students used in making their decisions on a project?

The faculty member plays the role of moderator and discussion leader in the classroom. He or she sets the pace for the class and provides the student with examples and challenges with respect to the subject matter. Some faculty also play the role of a Socratic antagonist, asking the students to defend their positions and decisions; in this role, the instructor is more interested in the "Why" behind an answer than in the answer itself. The goal is for the faculty member to serve as the facilitator of student learning and not to be viewed as the oracle of knowledge.

The faculty also use the Studio projects and even current events as examples to show the students how the material relates to the field of Software Engineering [3]. These create a continuous set of case study opportunities to show students how what they are learning in the classroom can be used to better understand and help to address issues encountered on real software projects.

Case studies from industry are common faculty teaching tools via which the students are challenged to provide analysis with respect to a particular topic of interest. The same case study may be used across a number of topics. The student might first be asked to evaluate the case study with respect to decision making as the key topic and then later the same case study might be used to reflect on estimation. When given such a task, the students present solutions, and the faculty question their reasoning, usually in open discussion.

Curricular Materials

We have discovered that maintaining a common representation for course materials helps to ensure that students more quickly learn "how to learn" in our program. All curricular materials for courses are provided on websites and managed using a learning management system similar to those used by other software engineering programs. One unique aspect of our program is the continuous review of courses and curricular materials by an executive committee of Software Engineering faculty to ensure that the courses and their materials continue to provide enduring principles and concepts to the students as the field evolves.

A possibly unique aspect of our program, is the use of student-generated curricular materials in the context of project work. Projects employ two major frameworks for student generated materials. MSE projects employ our Proposal Based Studio[2] which forces the declaration of the processes and tools the students have learned about in their courses and will use to solve the project problems. These are used as a guide for the mentors in evaluating the student's progress in the Studio. The idea of "say what you will do, and do what you say" underlies the materials that will be used to evaluate the real-world projects -- always a difficult task. The proposal is negotiated among the mentors, their student teams, and the customer. It enables a consistent assessment of projects across the multiple domains and environments encountered in the Software Engineering program. The students update their project proposals each semester to allow them and the mentors to adapt in light of increasing project experience. Thus, the measurement framework, processes, and approaches evolve as the project is changing, while the proposal remains an anchor for the project.

MSIT practicum projects also employ a student proposal created before the project begins. The proposal is similar to the normal project statement of work and like a Studio proposal, explains how the team plans to solve their problems. Unlike the Studio, this proposal specifically discusses how the team will attack the project problems with what they have learned in each of the five core courses and optionally in their electives. This practicum proposal

again acts as a framework against which the practicum project is assessed and enables the mentors to be consistent in their evaluation of across practicum projects.

We believe that having the students explain how they will use what they have learned to handle the uniqueness of a real-world project in proposals, is a key way in which students learn to apply our curriculum to real-world software engineering applications.

How Students Are Assessed

Individual student performance is difficult to assess in Software Engineering programs because the "lone wolf programmer" is a dying concept. Software Engineers are expected to work in teams almost from day one when they arrive in companies. In the curriculum we put the students into teams, early and often. This provides realism but poses challenges for individual assessment. A student can "hide" in a team and perform reasonably well as part of that team but never really master the material at the level we would like as faculty. To meet this challenge we have developed some techniques to facilitate faculty evaluations of individual students.

All student teams are required to do multiple presentations throughout the semester -- in the extreme, they must present every two weeks during our MSE "bootcamp"[7]. (Bootcamp is a 14 week discussion course of topics found to be useful in helping teams form quickly and understand what the MSE program is attempting to teach.) This represents seven presentations per semester for each team and every individual is expected to give one or more presentations. The students' grades depend in significant part on the quality of their presentations. Each subsequent course in the curriculum has a significant presentation component which provides additional opportunities (and requirements) to present.

Course work tries to balance group and individual evaluation. For example in an assignment, a team might be asked to develop a solution to a development problem, but then each individual team member is asked to reflect on how well the team solution worked, what they might have done differently, and how the team worked together to solve the problem. This individual reflection gives us an insight into how well the individual student is learning the material and is able to evaluate the work they are doing with respect to what they have learned. Reflection is a continuous component of assessment and an expectation for both individual students and for the student teams on projects. In some courses exams and quizzes are used to aid in individual evaluation.

All students also have individual meetings with their project mentors/coaches. These meetings provide a vehicle for evaluating how students are using the concepts taught in the classroom in their projects. The students are not evaluated against each other, but rather against the problems they have been presented in the project context and how they have addressed those problems. The key idea is that students are expected to try what they have learned and to be able to reflect on why it worked (or not) on their projects. The mentors spend significant time asking students why the team did what it did in light of what has been learned.

The faculty also use 360 degree peer reviews within teams to understand how team members perceive each other and themselves. This provides input into overall assessments but is not the sole contributor. In nearly all courses, a percentage of the grade is dependent upon the subjective assessment of the faculty member. This will usually influence a grade within one plus or minus grade level (which provides the faculty with an incentive to perform careful, defendable assessments of students' performance in a course).

All mentors meet regularly to discuss team progress in their projects and to calibrate grading of the teams across the projects. The "All Mentor" grading meeting follows the final presentations of all the teams at the end of the semester, grading is reviewed, and grades are assigned in a open forum with all faculty present. More globally, we discuss our assessments of students at an end-of-semester "Black Friday" meeting during which the performance and progress of all students are reviewed.

3. THE SILICON VALLEY PROGRAM

Originally, Carnegie Mellon's Silicon Valley Software Engineering program taught the same core content as the Pittsburgh program -- only in a different way using a completely project- and team-based approach tailored to the needs of its student body. The students are comprised almost entirely of working professionals attending graduate school part time. These students have typically been out of college for some time and approach traditional graduate study with considerable trepidation. Over time, the content as well as the delivery of the program has evolved to reflect the interests (and work situations) of our largely Silicon Valley student body. It emphasizes:

- Product development
- Application of software engineering principles to smaller projects with relatively short development cycles
- Agile methods
- Entrepreneurship.

The program prepares experienced developers for leadership roles in development groups and beyond. More specifically, our students are typically seeking to advance to a senior software engineer, architect, or project manager role; many hope to become entrepreneurs.

Our goals are to equip students with a broad range of knowledge and skills directly relevant to their professional practice and to impart facility at applying these skills to real-world problems. To achieve these goals, we have adopted a pedagogy based heavily on team-oriented projects, simulations, just-in-time tutorials, and industrial practicums. Our curriculum design and course delivery methods rely heavily on experience gained from the Pittsburgh program's success with the MSE Studio, analyses of emerging core requirements to train professional software engineers (e.g., [8]), and a body of cognitive science knowledge of how adults learn effectively.

The Software Engineering curriculum comprises six semesters of project-based courses. Working almost entirely in teams, students act as employees of a fictional company hired to architect, design, and implement or manage the development of a series of very different products on aggressive schedules. Within this framework, and with support from faculty members, students confront realistic technical and business problems, including conflicting requirements, limited resources, and challenges of team leadership. This *Story-Centered Curriculum* [9] fosters initiative, collaboration, leadership, and repeatable success.

The program offers two tracks -- Technical and Development Management -- to prepare students for a range of careers. The two tracks share a common core of software engineering concepts, methods, and practices. The actual courses taken depend on the track and the available electives for that track. The Technical track prepares graduates to effectively architect, design, develop, and deploy complex software systems. The Development Management track prepares graduates to manage projects, processes, and people, whether in-house or outsourced. Students learn the technical, business, and leadership skills required to successfully manage software projects throughout their complete lifecycle.

The common core of the curriculum comprises three courses taken during the first year: Foundations of Software Engineering, Requirements Engineering, and Architecture and Design. Foundations provides an end-to-end experience with agile software development in which students apply a slightly modified version of Extreme Programming to convert an existing application from a desktop version to a modern web-delivered application and to extend its functionality based on interaction and negotiation with product management stakeholders. In Requirements Engineering, students apply a variant of the Unified Process while eliciting, analyzing, and documenting requirements for a web-based social software application. In Architecture and Design, students research a range of architectural styles in a case study context and then go on to architect and prototype the previously specified social software application. Technical students take a range of required and elective technical courses in the second year, culminating in a real-world practicum project. Similarly, Development Management students take a range of required and elective software management courses; they can also opt to do a practicum project, but they have the option of taking additional software management courses instead. Our very applied course in Innovation and Entrepreneurship is a popular elective.

In addition to the primary subject matter of the courses, several "threads" are woven into the curriculum, providing regular opportunities to practice soft skills such as:

- teamwork, including virtual teamwork and the use of collaboration tools
- written communication
- presentation
- negotiation
- principled decision making
- conflict resolution
- working with people from different cultures
- self awareness and reflection.

Interestingly, in our surveys of alumni, most count these skills among the most valuable things they learned in the program.

We believe the Carnegie Mellon Silicon Valley Software Engineering Program to be unique. While other US and European schools offer software engineering education, none adopt as intense a learn-by doing approach with the goal of producing a transformative experience for practicing software professionals.

How Students Work and Learn

As noted earlier, nearly all student work is done in teams. Teamwork is fundamental to the program for several reasons, most notably:

- Virtually all real-world software projects are of a scope that requires significant teamwork
- Teamwork enables students to have the experience of completing a realistic project and producing a full range of authentic work products
- Students are highly motivated by being members of a high-performing team working on an intense project.

Students also do some individual work to ensure that each is learning and contributing, to broaden their knowledge, and to aid in student assessment. Most frequently, this work takes place in the form of "management briefings" in which individual students must produce short written memos on technologies, development methodologies, or decisions confronting their teams.

Teams are formed according to a number of criteria:

- Pre-existing knowledge and skills of each team member, gleaned from pre-admission interviews of each student and a self-assessment questionnaire (In subsequent semesters, teams are re-formed repeatedly based on faculty knowledge of students' strengths and weaknesses.)
- Balance, so that each team member has some relative strengths and weaknesses, making each member valuable and providing the potential for peer teaching
- Geographic location is also considered, but student teams are nearly always formed to include remote members making virtual teamwork a necessity
- Work schedules and style preferences are also considered.

Since virtual teams are the norm, significant time is devoted to jumpstarting high team performance. The program begins with a three-day orientation devoted largely to effective teamwork. Students work together face to face performing intense but enjoyable tasks to aid them in gelling as teams before those teams "go virtual" to carry on the work of the program. Before the beginnings of the third and fifth semesters, students must attend weekend-long "Gatherings" during which they engage in tasks and social events aimed at strengthening interpersonal and team connections and broadening them across the student body.

Teams are expected to self-organize to achieve the tasks that they are assigned. They are generally encouraged to adopt the roles of the Team Software Process [10], modifying them as appropriate. In particular, teams are encouraged to add a "learning manager" who coordinates team learning activities such as producing an explicit learning plan in addition to the work plan for each task, dividing responsibility for optional learning materials, and facilitating the team's discussions of readings and other learning resources.

A team has a faculty coach (not a teaching assistant) who assists the team in assigning roles, defining its own processes, and executing those processes effectively with appropriate monitoring. In addition to learning from faculty coaching, students learn from rich curricular materials indexed to their tasks, and they learn from responding to in-depth faculty feedback on their deliverables and revising those deliverables to improve mastery of targeted knowledge and skills. At the end of each project, they learn from reflection activities designed to promote generalization of their learning experiences. Finally, they perhaps learn most of all from each other by sharing a range of knowledge and professional experiences ranging from work at small start-ups to large aerospace companies.

Curricular Materials

Curricular materials are provided on a program website. Note however, that this is not eLearning, per se; the materials are supplementary to interaction with faculty and each other. Each course is divided into several tasks, each yielding deliverables for evaluation. The website provides teams with significant performance support for their tasks. In most courses, each task is assigned via a *simulated email from a "company executive,"* and follow-up emails convey additional scenario materials providing grist for a team's work. A *plan of attack* provides a skeletal work plan to assist the students in planning their work. *Tips and traps* provide expert heuristic advice on aspects of the task, especially pointing out subtle pitfalls which students should avoid. *Readings and other learning resources* are indexed to aspects of the task to direct students to material directly relevant to their contextualized learning needs and to establish the relevance of all such material in practice. Finally, a pre-submission *checklist* encourages students to self-check all deliverables against faculty-formulated grading criteria before final submission.

How Faculty Teach

Faculty provide several kinds of educational support in a Story-Centered Curriculum. In addition to supervising the running of a course, faculty play several instructional roles. Depending on the total course enrollment, all roles can be filled by a single faculty member, or multiple faculty might be involved.

Team Coach. Since the bulk of student work and learning is done in teams, the faculty role of team coach is preeminent. A team coach assists teams in developing an effective team process, helps resolve team issues, mentors students to use relevant materials and approaches effectively, and reviews early drafts of student deliverables. Coaches sometimes provide direct guidance such as a just-in-time mini-tutorial, but more often they model problem solving techniques and ask open-ended questions to lead the students to discover relevant knowledge and to solve problems themselves. Coaches typically meet with teams once per week and have frequent email and telephone follow-ups with individuals as well as the team as a whole. The coach's closeness to a team enables him or her to provide accurate input into the grading process regarding individual performance. At the end of each project, the coach also hosts a team reflection session to reinforce what was learned, discuss team process, and facilitate peer reviews. In large courses, different faculty members provide overall course supervision and coaching. When course size permits (usually 25 or fewer students), however, faculty play both roles, and students appreciate the instructional continuity.

Subject Matter Expert. The course supervisor (or lead instructor) is typically the primary subject matter expert for the course, but additional faculty or outside experts may be available as consultants to provide just in time tutorial instruction and to answer questions about technologies and methods that students might choose to explore in depth. As a result of student demand, all courses also have weekly "seminar sessions," involving the entire class, in which subject-matter expert faculty facilitate discussions of readings and topics of general interest; these sessions also sometimes feature just-in-time tutorials on knowledge and skills relevant to the students' immediate work.

Roleplayer. Depending on the nature of the simulated scenario, one or more faculty members will play fictional management roles to provide guidance, data, and informal information as grist for the students' work. Typically, such a faculty member will meet with student teams individually or during seminar sessions several times during the course, for example as the VP of Engineering or Marketing or the CEO. Having several distinct roleplayers allows students to encounter and deal with divergent opinions and, thus, to sharpen their analysis and negotiation skills. All student presentations are made to roleplaying faculty. These faculty members also provide appropriate contextualized instruction and suggest additional learning opportunities.

To summarize a key aspect of the discussion above, faculty employ a range of research-validated teaching strategies:

- Open-ended questioning to guide students to discover knowledge themselves [cf. 11]
- Cognitive Apprenticeship, especially modeling effective problem-solving approaches, typically in problem contexts analogous to the students' work [12]
- Just-in-time mini-tutorials whose content is *immediately* relevant to the students' work [cf. 13]
- encouraging peer learning.

Our teaching faculty are unique because each has significant real-world experience in large companies and/or entrepreneurial ventures, as well as traditional academic credentials and significant teaching experience. Our faculty might thus be regarded as a "clinical faculty" in the sense envisioned by the Harvard Business School symposium on business education [14].

How Students Are Assessed

Although students work in teams, individual grades are assigned at the end of each course. As in Pittsburgh, this can be challenging for Silicon Valley faculty members, but several mechanisms are employed to ensure that weaker students do not "hide in teams" and that stronger students receive credit for their higher performance.

Team Grades. The team's grades for the various deliverables are a starting point for assigning final grades, and the team grade typically contributes about 80% of each student's grade. An individual's grade can, thus, vary up to two letter grades from the team's grade; however, a range of one letter grade plus or minus is typical. We employ what might be called a "limited mastery" approach to team deliverables and assessment. Teams are encouraged to turn in draft work for in-depth feedback and have the opportunity to revise the work before it is graded.

Individual Work. Components of team deliverables are often attributable to individuals. Students are also required to produce individual work at regular intervals and to present regularly; furthermore, faculty may require individual work on an ad hoc basis, when it seems necessary to assess a particular student's performance.

Peer Review. Student teams are also required to complete a peer review at the conclusion of each course. Each student uses a structured instrument to assess the strengths and weaknesses of each team member, including him- or herself. Students are not penalized for accurately assessing personal weaknesses; instead, these become targeted areas for self-improvement.

Coach's Input. Finally the coach, who has spent many hours working with the team during the course, provides input. The supervising faculty member and coach look for a confluence of indicators when adjusting an individual's grade relative to the team's grade.

4. CONCLUSION

Consideration of these two Software Engineering master's degree programs should make it clear that, like Software Engineering itself, Software Engineering education at Carnegie Mellon continues to evolve as well as to adapt to different student populations and work contexts. That said, however, the curricula remain grounded in core concepts, and the demonstration of knowledge continues as the cornerstone on which professional education is built. While pedagogy may vary, standards are set consistently high to ensure the highest quality education is delivered, and the highest quality graduates are produced.

To date, the main campus program has graduated 403 MSE students between the foreign campuses and the Pittsburgh campus program. We have also graduated 201 MSIT-SE students. Versions of the main campus program are currently being offered in Korea, India, and Portugal.

The Silicon Valley Campus has graduated 236 students with an MS in Software Engineering -- 171 in the Technical Track and 65 in the Development Management Track. We have also graduated 104 students with an MS in Software Management. (The MS in Software Management is distinct from the Development Management track of Software Engineering. It attracts mid-career

professionals aspiring to senior management and the curriculum is not discussed in this paper.)

For the past two years, we have surveyed alumni of all Carnegie Mellon Silicon Valley programs to ascertain the career value they attribute to their graduate education. (No comparison data for other programs are available.) In September 2008, 45 of 236 Software Engineering alumni completed the survey. Eighty-seven percent of respondents believe the program gave them a competitive advantage in their careers relative to their corporate peers. Many of our students have been promoted: 41% during the program and 45% after graduation; 82% changed jobs (either within their company or by moving to a new company).

Our students have also seen significant salary increases:

- 26% of respondents, greater than 40%
- 13% of respondents, 21-40%
- 33% of respondents, 11-20%
- 28% of respondents, less than 10%.

As noted earlier, most students tended to value soft skills, such as teamwork and effective communication, more than technical skills in hindsight. Eighty-three percent of respondents included one or more specific soft skills among the most important three things they learned. Proficiency in technical skills is assumed of graduates from top graduate programs; facility in soft skills is a key differentiator -- and one that is sometimes sorely lacking in graduates of traditional programs.

Finally, 87% of respondents would recommend Carnegie Mellon Silicon Valley to friends with interests similar to their own. Rather than ending this discussion with dry statistics, however, let us end it by letting some of our students speak for themselves:

> *The program's learn-by-doing curriculum mimics the way the software industry works in the real world. The faculty guided us through software processes, assigning work that consisted of writing code, completing projects, leading teams, and negotiating with stakeholders about requirements and deliverables. The program exposed me to a variety of techniques and methodologies for developing software, which I really appreciated, since at work I am only exposed to my company's process. However, the program truly exceeded my expectations in how it taught me the importance of team building and soft skills. Understanding the importance of these skills and honing them throughout my two years has helped me not only professionally but personally as well.*
>
> — Silicon Valley MSSE 2008 graduate

> *I am already taking away a lot from my schoolwork and applying it to my job because I can leverage it right away. What I learn on Monday, I can apply on Wednesday.*
>
> — a student early in the Silicon Valley program

> *I have used every concept from the curriculum in my daily work as a senior developer on challenging software projects.*
>
> — Pittsburgh MSIT-SE '05
> Staff Software Engineer, Lockheed Martin

> *The MSE program pushed me to my limits … and beyond. Far more than the tools, techniques and methods I learned at Carnegie Mellon, it is the confidence that I can thrive in the demanding business world of software engineering that has served me best since graduating.*
>
> — Pittsburgh MSE '95
> Director, Research and Development
> Misys Healthcare Systems Homecare Business Unit

> *The relationships you build from the mentoring program will last much longer than the school year, and are much more valuable than just an education. The technical and personal skills gained from the MSE program enabled me to influence positive change and effective organizational transformation.*
>
> — Pittsburgh MSE'02
> Senior IT Specialist, IBM-Rational Software

> *The MSE program is the perfect combination of management and technology for anyone in the software engineering field. What adds to the experience is the availability of so many instructors who are widely recognized in industry and academia. It's cool to be sitting in class listening to the people who wrote your college textbooks.*
>
> — Pittsburgh MSE'01
> Senior Member, SEI Technical Staff

Despite these challenging times, our graduates appear to remain in very high demand, and every year we receive acknowledgment from our alumni of the practical value of their Carnegie Mellon Software Engineering education. The practical nature of what and how we teach ensures the students will have immediate professional relevance, and the general skills they gain ensure that they will continue to maintain that relevance throughout their careers in this rapidly changing field.

5. REFERENCES

[1] D. Garlan , D.P. Gluch , J.E. Tomayko, Agents of Change: Educating Software Engineering Leaders, **Computer**, v.30 n.11, November 1997, p.59-65.

[2] D. Root, M. Rosso-Llopart, G. Taran, "Proposal Based Studio Projects: How to Avoid Producing "Cookie Cutter" Software Engineers," **CSEE&T 2008**, pp. 145-151.

[3] G. Taran, D. Root, M. Rosso-Llopart, "Continuing Challenges in Selecting Industry Projects for Academic Credit: Points to Consider and Pitfalls to Avoid," **CSEE&T 2008**, pp. 163-170.

[4] G. Taran, "Using Games in Software Engineering Education to Teach Risk Management,". **CSEE&T 2007**, pp. 211-220.

[5] M.C. Paulk, B. Curtis, M.B. Chrissis, C.V. Weber, **Capability Maturity Model for Software, Version 1.1**, Technical Report, CMU/SEI-93-TR-024, ESC-TR-93-177, February 1993.

[6] W. Humphrey, **Managing the Software Process**, Reading, MA: Addison-Wesley, 1990.

[7] D. Root, M. Rosso-Llopart, G. Taran, "Key Software Engineering Concepts for Project Success: The Use of "Boot Camp" to Establish Successful Software Projects," **CSEE&T 2007**, pp. 203-210

[8] M. Shaw (editor), **Software Engineering for the 21st Century: A basis for rethinking the curriculum**, Technical Report CMU-ISRI-05-108, Carnegie Mellon University, Institute for Software Research International, 2005.

[9] R.C. Schank, **Making Minds Less Well Educated than Our Own**, Mahwah, NJ: Lawrence Erlbaum Associates, 2004.

[10] W. Humphrey, **Introduction to the Team Software Process**, Reading, MA: Addison-Wesley,1999.

[11] Staff, Harvard Business School. **Case Method Teaching**, Report 9-581-058. November 6, 1998.

[12] A .Collins, J.S. Brown, and A. Holum, "Cognitive Apprenticeship: Making Thinking Visible," **American Educator**, Winter 1991.

[13] J.D. Bransford and D.L. Schwartz, Rethinking Transfer: A Simple Proposal With
Multiple Implications, **Review of Research in Education**, 3(24), 2001, pp. 61-100.

[14] Staff, Harvard Business School Discusses Future of the MBA, **HBS Alumni Bulletin** (http://hbswk.hbs.edu/item/6053.html) November 24, 2008.