Developing Software Engineering Leaders at Carnegie Mellon Silicon Valley

Ray Bareiss
Carnegie Mellon University
Silicon Valley Campus
MS 23-11
Moffett Field, CA 94035
011-650-335-2801
ray.bareiss@sv.cmu.edu

Todd Sedano
Carnegie Mellon University
Silicon Valley Campus
MS 23-11
Moffett Field, CA 94035
011-650-335-2812
todd.sedano@sv.cmu.edu

Abstract

The Carnegie Mellon's Silicon Valley Campus offers a master's degree in Software Engineering, with technical and development management tracks, targeted at working software professionals in Silicon Valley. We believe the program to be unique in that it is entirely team-based and project-centered. Students learn by doing as they are coached just in time by faculty in the context of their work on authentic projects, and they are evaluated based on what they produce. In response to our interactions with an industry characterized by innovation and short project development timelines, the program evolved from one focused on "high ceremony" processes to one focused on agile software development methodologies. Student satisfaction is high: Eighty-seven percent of our alumni believe that the program has given them a competitive advantage with respect to their professional peers, and their promotion and salary histories bear out this belief.

1. Introduction

Students typically enroll in a professional master's program to gain practical knowledge and skills, which enhance their career prospects, and to grow their professional networks. These students have typically been out of college for some time and approach traditional graduate study with considerable trepidation. They are used to the rhythm and evaluation criteria of the workplace. Standardized entrance exams (e.g., the Graduate Record Examination) stress them out. The thought of returning to the artificial evaluation of academic grading reminds them that the skills and incentives of the workplace are all too often different than those of academia. From the working professionals' point of view, the most useful graduate education would align with their professional work, and it would provide an enhanced, guided and practical version of the life-long learning they must do on the job.

The overarching goal of Carnegie Mellon Silicon Valley's professional master's programs is to provide a *transformative educational experience* to our students, one that effects a fundamental change in the way that students behave as software engineers, not just in the educational context, but more importantly in their professional work as well:

- During the course of our program, students are equipped with a broad range of knowledge and skills directly relevant to their professional practice, and they gain facility at applying these skills to real-world problems
- Students' decision-making processes significantly improve as they learn principled decision-making frameworks and how to apply such frameworks logically
- Students learn to express their ideas clearly and persuasively, and they become able to negotiate effectively and with authority
- Students become adept at working effectively in teams, perhaps the key skill in software development of any scale
- Students become effective self-directed learners, which is essential in a field in which some put the typical half-life of knowledge at approximately two years.

Juxtaposing this overarching goal of professional transformation with Cognitive Science research on how people think and learn (see, e.g., [1]) suggests the properties of an effective educational approach:

- The approach should center on active problem solving to promote the acquisition of usable knowledge rather than a collection of memorized facts

- The approach should situate learning in a realistic context, highly similar to the environment in which students are expected to apply the knowledge, thus promoting transfer
- Tasks in the realistic context should ensure that the targeted knowledge and skills are required for successful problem solving
- Instruction should be "just in time," when problem solving tasks provide a context for processing new information and storing it in the learner's memory in a useful form with appropriate indices
- Knowledge and skills should be taught holistically, as they will be applied, rather than separated into academic silos
- The learning experience should equip students with the fundamental skill of self-directed learning.

In response, we have adopted a pedagogy based heavily on team-oriented projects, simulations, just-in-time coaching and tutorials, and industrial practicums, delivering our courses as Story-Centered Curricula [2]. Our curriculum design and course delivery methods rely heavily on experience gained from Carnegie Mellon's successful Studio-based experience with the Master's of Software Engineering program in Pittsburgh [3], several analyses of emerging core requirements to train professional software engineers (e.g., [4], [5]), and a growing body of cognitive science knowledge of how adults learn effectively. We have heavily modified the Carnegie Mellon Pittsburgh delivery model and the content of the program to better match the style of fast-paced software engineering practiced in Silicon Valley. We have instantiated this educational approach to offer a professional master's program in Software Engineering with tracks in Technical Software Engineering and Development Management, as well as a sister program in Software Management which addresses the needs of senior professionals.

When we launched the Software Engineering program in 2002, our initial strategy was to play to the strengths of Carnegie Mellon's software engineering education, focusing on "the development of business-critical, network-centric, software-intensive systems of systems (employing commercial off-the-shelf components, outsourcing, and internal development) requiring predictability and quality in their development and operation" (language from our first brochure). Such systems are typically developed by companies employing relatively high ceremony development processes and measuring their organizational capabilities via the Capability-Maturity Models of Carnegie Mellon's Software Engineering Institute [6]. We also emphasized use of the Personal Software Process at the individual level [7].

Not surprisingly, our initial partners in launching this degree program were large companies, especially aerospace and defense companies. Nearly 50% of our initial class of students came from such companies. The remainder came primarily from Silicon Valley companies possessing quite a different world view. These companies, and thus their employees, are product focused, generally engage in smaller projects, favor agile development methodologies over high-ceremony processes, measure development cycles in weeks instead of years, and are part of the entrepreneurial culture of Silicon Valley.

During the early years of the program, we experienced a significant tension between the needs of these two constituencies. Enrollment trends resolved this tension. The number of Silicon Valley professionals enrolling in our program increased, while the enrollment of aerospace and defense professionals decreased. Only 16% of the graduating class of 2009 work for aerospace or defense companies. To meet the needs of our students, our program has evolved to feature a strong product development focus, agile development within short development cycles, and entrepreneurship. During this evolution, we have retained the dual emphasis on business and technical skills. And over time, the tension is diminishing; in fact, our students from aerospace, defense, and other large companies increasingly share these interests as well and have established a history of taking agile development ideas back to their large projects.

We believe this 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.

2. Details of the Software Engineering Program

The Master's of Science in Software Engineering (MSSE) degree program has two tracks which mirror the central career choice of each software engineer: whether to stay technical or to move to management. The two tracks prepare students for different career paths while providing a common core of software engineering knowledge to all students. The Technical track is for students who want to remain close to writing software, most often aspiring to be technical leads or software architects; the track teaches students to effectively architect, design, develop, and deploy complex software systems. The Development Management track is for students who want to move

from hands-on development to technical management, aspiring to be project or functional managers; the track focuses on managing processes, people, and projects, whether in-house or outsourced.

Typical applicants have several years of work experience, a computer science or related degree, and some team-based software development experience. We expect incoming students to have completed lower division CS courses or to have the equivalent practical experience. When evaluating a student, we weigh a combination of industry experience and academic preparation: thus, a recent computer science graduate might be given the same consideration as an applicant with 10 years of software development experience and an undergraduate degree, say, in mechanical engineering. For our full time program, one third are domestic students and two thirds are international students. For our part time program, virtually all are domestic, and thirty to forty percent of the students are remote; many of these are employees of aerospace companies. Local students are drawn from a large range of Silicon Valley companies, comprising early-stage start-ups to IBM, HP, Oracle, and Google – and everything in between. To encourage a strong sense of community, all students, whether remote or local, are required to come to campus for a three-day orientation, on campus "Gatherings" before the third and fifth semesters, and for graduation. All students are also strongly encouraged to attend commencement in Pittsburgh.

2.1 Example: The Foundations of Software Engineering

The Software Engineering curriculum begins and ends with end-to-end software development experiences. *Foundations of Software Engineering*, the first course in our program, provides students with an intensive software development experience with significant scaffolding to ensure that they follow industrial *and team* best practices. The scope of student projects differs as they renegotiate and refine project requirements throughout the course, but all students share a baseline experience with respect to one software method, currently a slightly modified form of Extreme Programming. At the end of the program, a real-world Practicum project provides no scaffolding as students decide for themselves which practices to follow given unique customer project requirements and constraints. This capstone project allows the student to demonstrate mastery over software engineering concepts and aids in the final transition of knowledge and skills from academia to professional practice.

The overall objectives of *Foundations* are to establish a baseline of principled software development skills and to prepare the student for the remainder of the program. Each student comes to recognize individual strengths and weaknesses and establishes targets for improvement during the rest of the program. The student also becomes familiar with our learn-by-doing pedagogy and our philosophy of effective team-based work, as well as a selection of foundational elements from computer science and software process.

Teams of three to four students "work for" ND System Solutions, a small fictional company which is developing a tool for software project definition, effort estimation using historical data, and effort tracking. As in industry, the faculty assign students to teams and hold a project kickoff describing the project, their stakeholders, milestone deadlines, key constraints, and available resources. In addition to covering all the key concepts found in an industrial kickoff, the faculty also place significant emphasis on how students will learn, specific learning objectives, working with faculty, working effectively in teams, *et cetera*. A faculty member plays the role of Vice President of Engineering (VP), monitoring the team's progress, and providing coaching and mentoring. As part of the scaffolding, we provide students with a high-level project plan, but student teams manage the specifics of the project plan as they implement the requirements and respond to customer feedback throughout the project. There are no formal tests during the course; students are graded on the work they produce, taking into account both team and individual accomplishments.

The *Foundations of Software Engineering* course originally taught a high ceremony approach to software development. However, as Silicon Valley companies have embraced agile methods, we have changed our course to teach agile practices emphasizing iterative development, frequent customer interaction, adaptability to change, agile estimation, continuous builds, and a releasable product at the end of each iteration. The faculty selected Extreme Programming as the agile method to teach because:

It is an agile method reflective of the Silicon Valley development culture

- It has a strong emphasis on engineering practices whereas many agile methods focus on process practices
- While student interest is high, many industry managers are reluctant to allow their employees to try aspects of Extreme Programming such as test-driven development or paired programming on the job.

The course provides a safe environment to allow students to try new ideas, and many become strong

advocates by the end of the course. While the VP encourages students to follow Extreme Programming faithfully, students need to customize the method since many of them are on distributed teams and all have limited access to the customer.

The 14 week course is divided as follows:

- Start-up: one week to sort out tools, technology decisions, team processes, begin the readings, and determine how the team will capture requirements in Iteration 0. The team deliverable is a team requirements template. Each individual also delivers a briefing on Extreme Programming.
- Iteration 0: three week requirements gathering sprint during which students elicit, refine, and prioritize requirements while prototyping and, thus, learning the technology used in the course. Team deliverables are prioritized story cards, a user interface specification, a data dictionary, and iteration planning meeting minutes.
- Iterations 1-3: three week implementation sprints culminating in a final presentation and a working product. Team deliverables are the product (including test cases, code, and design documents), weekly cycle meeting minutes, iteration planning meeting minutes, and a final presentation. Each individual also delivers a Myers Briggs briefing and peer evaluations.
- Retrospective: a one week wrap-up during which the team conducts a postmortem and reflects on what they have learned. The team deliverable is a retrospective report on the project.

Throughout the course, the Vice President of Engineering (i.e., the supervising faculty member) meets weekly with each student team to review their work and check on their progress as a team.

The VP provides the voice of the customer. The VP is not a de facto manager, telling the team what to do but rather helps the team to decide its own direction and to move forward effectively; in many situations, this is operationalized as open-ended questioning as the team considers decisions, cf., [8], [9]. Through interaction with the VP, students learn agile requirements elicitation and prioritization, test-driven development, paired programming, continuous integration, agile project metrics and reporting mechanisms, how to analyze test coverage, how to release a project and how to give a demo presentation. A strong emphasis in this interaction is on balancing agility and discipline appropriately to achieve the goals of the project. The faculty member playing the role of VP must be an expert software engineering practitioner.

The VP serves as the team's coach. We emphasize high-performance teamwork at every opportunity. During an intensive three-day orientation for new students, we introduce the students to their teams, engage them in team-building exercises, and provide conceptual tools such as Situational Leadership [10] and Tuckman's model of team development [11]. We use the Myers-Briggs personality sorter as a mechanism to help each student to realize how their teammates are different than themselves. A significant portion of faculty interaction revolves around team development. The VP also helps the team to define its learning processes, to plan its work, and to accomplish its goals effectively. In team meetings, the VP uses the questions such as "What is working well for the team?", "What can the team improve upon?", and "What is the stress level of each team member on a scale of one to ten?" to provide opportunities for "micro-reflections" on team development at least once a week.

Throughout the semester, students also attend weekly faculty-led discussion sessions and practice sessions. In the discussion sessions, students explore the reading topics much like a graduate seminar by participating in faculty-facilitated discussions. In the practice sessions, we address common skills deficits, including the use of new development technologies such as Ruby on Rails, version control, test driven development, and writing and presentation skills. Such sessions combine small amounts of faculty presentation with extensive hands-on practice by the students.

2.2 The Remainder of the Curriculum

During the first year, all students take the curriculum's core courses. The core courses address most of the key skills needed by software engineers. *Foundations of Software Engineering* has already been discussed in detail. In *Requirements Engineering* students experience the requirements process in depth; they use techniques drawn from Human-Computer Interaction to better understand and document the requirements for a software product. In *Architecture and Design*, students are guided through an examination of many architectural styles, looking at the strengths and limitations of each and determining how they can best be applied; the students create two distinct architectures for the product specified during Requirements Engineering and use ATAM to evaluate how well each architecture satisfies the product's quality attributes.

During the second year, the technical track courses prepare students for a technical career path. In

Avoiding Software Project Failures, students are guided through several case studies of failed software projects to understand the root causes of costly mistakes. In *Metrics for Software Engineers*, students analyze and propose metrics initiatives for software development teams in agile and traditional organizations, focusing on metrics designed by engineers for engineers. In the *Practicum Project*, teams apply what they have learned to a real-world problem; working with a client, the team negotiates plans, schedules, and deliverables, and then develops their final product while adhering to high standards for software engineering approaches, accountability, and teamwork.

The development management track courses prepare students for a project or development management career path. In *Elements of Software Management*, students are guided to assess real software businesses from marketing, business strategy, financial, and overall business perspectives, applying fundamental methods, models, and frameworks. In *Metrics for Software Managers*, students analyze and propose metrics initiatives for fictional software organizations with specific software management problems, aligning the initiatives with business and stakeholder goals. In *Project and Process Management*, students explore the component processes of agile and traditional software development; they then propose a hybrid development methodology for a particular project and manage the project through a series of critical events in simulation. In *Managing Software Professionals*, students address a range of "people issues," in simulation, related to hiring, retention, and dismissal of employees, as well as cultural considerations of managing a diverse team.

Electives are chosen purely on the basis on student interest. Popular electives include *Innovation and Entrepreneurship*, *Human-Computer Interaction*, *Open Source Software*, and *Managing Outsourced Development*. Development management students (for whom a capstone project is not required) can opt to do a *Practicum Project* in lieu of electives.

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 facilitating collaboration and virtual, distributed teamwork
- written and verbal communication, including making effective presentations
- effective meetings
- negotiation
- conflict resolution and working with people from different cultures
- principled decision making
- self awareness and reflection.

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

3. Overview of Instruction at Carnegie Mellon Silicon Valley

3.1 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 development methodologies, decisions confronting their teams, or concepts the faculty want to verify that every student understands.

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. 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 sometimes include remote members requiring students to learn virtual teamwork skills.

Teams are expected to self-organize to achieve the tasks that they are assigned. In addition to roles related to managing and carrying out their work, 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.

Each 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.

3.2 Curricular Materials

Curricular materials are provided on a program website. Note however, that this is not eLearning, per se; the materials are supplementary to faculty and student interaction. Each course is divided into several tasks, each yielding authentic 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.

3.3 How Faculty Teach

Faculty provide several kinds of educational support. 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 facilitates 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 and grading 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. 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. [10], [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.

We recently surveyed our students' and graduates' attitudes towards our teaching methods. 77% believe that our teaching methods are more effective, in general, than others they have experienced at the university level (12% are neutral and 11% believe they are worse).

Student comments accompanying the survey responses suggest that faculty are sometimes too "Socratic" in their approach to teaching, tending to turn questions back to the students rather than providing answers and expressing their own opinions. Students would also like faculty to lay out theoretical frameworks when new disciplines are introduced rather than focusing entirely on coaching in reaction to team and individual decisions. That said, however, only three of the 120 students and alumni who responded suggested that formal lectures would improve the educational experience.

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].

3.4 How Students Are Assessed

Although students work in teams, individual grades are assigned at the end of each course. This can be challenging for 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 one 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 himself 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.

5. Outcomes

Carnegie Mellon Silicon Valley 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

5. Acknowledgments

Our thanks to Martin Griss and the rest of the Carnegie Mellon Silicon Valley Software Engineering faculty, past and present, for their energy and innovation that have made this program a success.

6. References

[1] J.D. Bransford, A.L, Brown, and R.R Cocking (Eds.), *How People Learn: Brain, Mind, Experience, and School.* National Academy Press, Washington, DC, 2000.

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

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

[4] M. Shaw (Ed.), *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, Pittsburgh, PA, 2005.

[5] A. Pyster (Ed.), *Graduate Software Engineering Reference Curriculum (GSwRC),* Stevens Institute of Technology, Hoboken, NJ, Version 0.50, October 2008.

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

[7] W.S. Humphrey, *Introduction to the Personal Software Process,* Addison-Wesley, Reading, MA, 1996.

[8] C.E. Hmelo-Silver, and H.S. Barrows, Goals and Strategies of a Problem-Based Learning Facilitator. *Interdisciplinary Journal of Problem-Based Learning 1,* 1 (Spring 2006), 21-39.

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

[10] K. Blanchard, *The One Minute Manager Builds High Performing Teams,* William Morrow, New York, NY, 2000.

[11] Wikipedia.org, *Forming – Storming – Norming - Performing,* http://en.wikipedia.org/wiki/Tuckman_Model, June 22, 2009.

[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