An Exploration of Knowledge and Skills Transfer from a Formal Software Engineering Curriculum to a Capstone Practicum Project

Ray Bareiss and Edward Katz
Carnegie Mellon Silicon Valley
{ray.bareiss,ed.katz}@sv.cmu.edu

Abstract

Students at Carnegie Mellon Silicon Valley complete a team-based practicum project for an industrial sponsor as the capstone of their master's education in software engineering. Over time, the faculty member who typically serves as advisor for such projects has been disturbed by the failure of several student teams to transfer some relevant knowledge and skills from the formal curriculum to the relatively unstructured practicum project environment. We conducted a survey of all 2010 software engineering students to ascertain the most significant self-reported shortcomings. This paper presents the survey data and then discusses the results in terms of a theory of transfer; as part of this discussion recent and possible future changes to instruction are identified.

1. Introduction

Carnegie Mellon Silicon Valley provides a unique master's program in software engineering targeting both working professionals, who complete the program part time over two years, and recent graduates, who complete it full time in one year. The program aligns with the emerging Silicon Valley approach to software engineering which is agile, product focused, entrepreneurial, and tends to focus on smaller projects completed on short timelines. The program features a unique team-based and project-centered pedagogy in which students *learn by doing*. They work on realistic projects, are coached by faculty to learn just in time, and are evaluated based on the work they produce.

The program is bracketed by end-to-end software development experiences. The beginning *Foundations of Software Engineering* course (described in a later section of this paper) involves completing a faculty-provided project which is realistic but designed with pedagogical intent; its completion requires particular knowledge and skills. The final *Practicum Project* is a real-world development project undertaken for a client; it provides a capstone experience in which students are expected to transfer the knowledge and skills learned over the course of the program to this authentic final project.

There is a large body of psychological literature on transfer, the ability to recall and apply knowledge and skills learned in one context in another (see, e.g., [1,2]).

> Transfer of learning occurs when learning in one context enhances (positive transfer) or undermines (negative transfer) a related performance in another context. Transfer includes near transfer (to closely related contexts and performances) and far transfer (to rather different contexts and performances) [1].

Generally, the results are disappointing [3,2]. Transfer is far less reliable than educators would hope and generally becomes worse as the similarity between the learning and application contexts decreases. The premise of our educational approach is to create a context for learning software engineering knowledge and skills that is highly similar to the work context in which they will be applied (cf. "hugging" in [4]). While self report by our students suggests this is largely the case, we have been disappointed in some students' abilities to transfer knowledge and skills from the formal curriculum to the relatively unstructured practicum project.

This paper describes our initial observation and exploration of this phenomenon. It begins with a brief overview of our formal curriculum, highlighting key knowledge and skills. It then describes observations that caused us to undertake this investigation. Next, the paper discusses the results of a survey of 2010 practicum students. Finally, it provides our initial thoughts on how to introduce more structure to the practicum experience to improve transfer.

2. Review of the "formal" software engineering curriculum

Many courses in our curriculum provide knowledge and skills that should flow directly into the students' practicum work. (For a more complete discussion of the program, see [5] and [6]).

The program begins with a semester long *Foundations of Software Engineering* course to which students are each expected to devote 20-25 hours per week. The course is team-based and project-centered, and is intended to provide students with an end-to-end experience in agile software development. Other key objectives include establishing a baseline of software development skills and acquainting the student with the learn-by-doing pedagogy which is the hallmark of our program.

Foundations, and indeed the program as a whole, has evolved an overall focus on agile development, which reflects Silicon Valley's embrace of agile methods. The course emphasizes iterative development, frequent customer interaction, agile estimation, writing test cases before writing code, continuous builds, adaptability to change, and a releasable product at the end of each iteration. The faculty has selected Extreme Programming as the development paradigm to be taught because of its strong emphasis on engineering practices. The course provides an environment to try out new ideas, and many students become strong advocates for implementing these ideas at their employers.

In addition to their project work, students attend weekly faculty-led discussion and practice sessions. Discussion sessions explore reading topics much like any graduate seminar. Practice sessions address skills deficits and teach technologies likely to be new to the students, such as Ruby on Rails, test-driven development, pair programming, continuous integration, and version control, as well as strengthening the students' "soft skills."

Later in the program, students take a course in *Metrics for Software Engineers*. The course, which is also team-based and project-centered, focuses on metrics for quality and productivity, as well as change management of metrics initiatives. The quality portion of the course provides opportunities to practice test-driven development, defect injection, and formal code reviews – all involving extensive data collection and analysis. The productivity portion provides opportunities to practice project estimation, including the use of historical data, and various approaches to progress tracking.

Other courses particularly relevant to the students' practicum work are *Requirements Engineering*, *Software Architecture*, and an *Introduction to Human-Computer Interaction* elective. Of particular relevance to later discussion, both the Requirements and Human-Computer Interaction courses place significant emphasis on "low-ceremony" usability testing.

In additional to technical content, a number of soft skills threads run throughout the curriculum: High-performance teamwork is heavily emphasized as are effective meeting practices, communication, negotiation, conflict resolution, and self-awareness and reflection.

3. The practicum project: Overview and issues

3.1 Overview

The *Software Engineering Practicum* [7] provides a context for students to apply their recently acquired skills and knowledge in a real-world setting, working on a software project for an actual client. Besides the primary benefits of exercising and extending skills and knowledge, working on such a project can have the secondary benefits of learning how various companies carry out projects and, for some, obtaining offers of post-graduation employment.

A client may want to participate in our software engineering practicum for a variety of reasons. Most commonly, the client has a small project but has insufficient resources to pursue it internally. For example, this might be to try out a software tool or to develop a prototype application in a short timeframe. Supporting a software engineering student team can be a cost-effective way to try out a new tool or technology and apply it to a particular prototype project while minimally taxing the client's internal resources. Nearly all of these projects are new development rather than continuation of work on existing projects.

A significant advantage to using a student practicum team is that they essentially function as a high-quality contract software development team. The team is already gelled, knowledgeable, and prepared to make rapid progress on the client's project. This is in sharp contrast to the client taking on student interns, who often have widely varying backgrounds and skills. Generally, interns also have little experience in professional team-based software development and require considerably more direction and oversight.

The first step in qualifying a project, after initial client contact and discussion, is the project proposal submission. A potential client submits a written proposal for each candidate project, following a set of well-defined authoring guidelines. The guidelines include specification of expected student knowledge and skill levels, major project goals, a suggested high-level project roadmap, expected major deliverables, known obstacles and risks, and measures by which success will be judged.

Each submitted proposal is reviewed by a committee of software engineering faculty to ensure its appropriateness as a Practicum project. Students then receive all acceptable proposals to review, enabling them to ascertain their initial interest and to note any questions they have. Approximately a week later, a Practicum Project Fair is held during which the proposal authors, other key project stakeholders, and the students meet face to face. This provides an opportunity for the students to get to know the clients and to ask clarifying questions about the proposals. It is in the students' interests to understand each project

thoroughly because they will be asked to express project preferences. In doing so, it especially helps students to have some personal familiarity with potential clients; client contact information is provided to enable students to follow up directly with any additional questions.

Teams are assigned to projects based primarily (but not exclusively) on students' individual preferences. A short time after the Fair, each student is asked to submit a ranked ordering of the projects from most to least desirable. Because these are all substantial projects, a team is typically required to have a minimum of three students. Faculty attempt to give each student his or her preferred choice, subject to the team size constraint and an appropriate balance of skills within the team. Because of these constraints, not every student can be assigned his or her first choice; we try hard never to assign a student to a project lower than his or her third choice.

The first meeting of the Practicum course is a boot camp without clients in attendance. The bootcamp presents the faculty's expectations and provides the pragmatic advice on dealing with a real customer to produce the expected deliverables. In particular, possible styles of client interaction are discussed. A client could interact with the team very closely as an active project participant. (This type of client tends to be a software professional.) Or the client could remain at a distance leaving most decisions to the team. (This type of client tends not to be a software professional.) Students must be prepared to deal with either situation and often do not know which they will experience until they begin working with the client.

After the bootcamp, each student team contacts their client directly to arrange an initial meeting, which the faculty advisor attends only as an observer. One of the team's primary goals is to try to ascertain the client's style and to plan how to best work together.

The client is expected to meet with the practicum team at least once a week. During these meetings, the client and team meet directly to agree on deliverables, milestones, schedules, et cetera. On the periphery, the faculty advisor observes the meetings to ensure that the team is actively engaging with the client and that the client is behaving appropriately and making reasonable requests. The faculty advisor also meets with the team separately to provide advice on client interaction, project organization, and, sometimes, subject matter expertise.

Teams are expected to self-organize and decide on team roles. Often these roles are rotated during the course of the project. The team is expected to select and tailor an appropriate development methodology to follow given the nature of the project. Often the client is not software engineering savvy, so the team is also expected to proactively define the necessary software engineering processes, select tools, et cetera independent of the client.

Client deliverables are determined via negotiation with the client. All such deliverables must typically be delivered by the end of the semester, but in rare cases, agreements are made with the client to deliver some work after the end of the semester.

In addition to client deliverables, the Practicum requires a combination of team and individual academic deliverables which are graded by the faculty advisor:

- A statement of work
- A final public project presentation
- An individual reflection report.

Each student's grade is based on a mix of team and individual deliverables. The client's assessment dominates, counting for 50% of the grade at the time of the study. Team and

individual deliverables and the faculty advisor's assessment of the team's project management performance make up the other 50%. (Changes are discussed in Section 5.)

3.2 Issues Observed

Over the past few years, the faculty advisor who typically advises Practicum teams has observed following issues while working with students:

- Poor testing/no test suites – Most clients are eager to see early working demonstrations. Student teams frequently respond by jumping right into coding the client's application rather than engaging in test-driven (or test-first) development as they have been taught.
- Over concern with satisfying the client – Teams often allow themselves to be guided almost completely by the client and ask the client to make all significant project decisions in the belief that "local" client satisfaction is paramount instead of maintaining the team's role as a working partner whose influence on decision making can have a large bearing on the ultimate success of the project.
- Ineffective teams – Under the pressure to deliver, students sometimes forget they are a team and lapse into working as a group of individuals; differentiated team roles disappear, and cohesion, communication, and productivity are significantly impacted.
- The misperception that agile means no documentation – Because agile methods generally value working code over documentation, student teams, under pressure to deliver to satisfy a client, either ignore the need for documentation or treat it as an afterthought. This can have two related negative consequences: It leaves the client with only source code which the client's organization may not be able to make further use of. Related to this, some clients request follow-on practicum teams to extend the project, and like the client's own people, the second team has little basis for understanding the initial team's work.
- No notion of how the project is doing and, in particular, no use of metrics – A team may be lulled into believing it is making satisfactory progress based on off-the-cuff, subjective feedback from the client only to come up short at the end of the project. Most teams do not think to use productivity metrics until the faculty advisor asks to see them (usually after the project is underway).
- No clear measures of project quality – Teams frequently depend on the customer's subjective assessment as their sole measure of project quality and don't think to implement quality assurance procedures and metrics.

4. A survey of 2010 software engineering practicum students

The 23 practicum students – 13 part time and 10 full time – in our Software Engineering Technical Track were asked to complete an anonymous survey of the practices employed during their practicum projects. Although students worked in teams, they were asked to respond to the survey individually to maximize the perspectives captured. Fourteen students responded: seven of the 13 part-time students who completed the practicum during Spring 2010 and seven of the 10 full-time students who completed the practicum during Summer 2010.

4.1. General questions

These six questions elicited free text input, and a student could give multiple answers. Student responses are categorized for purposes of this discussion.

1. How do you know if your practicum project is on track?

Regular client meetings as checkpoints	6
Scrum project management techniques (e.g., iteration planning, daily stand-up meetings)	6
Tracking progress against a traditional plan	2

The faculty's preferred answer to this question would involve the use of quantitative metrics, ideally those learned in Metrics for Software Engineers, as well as qualitative measures. Only three of the fourteen responses mentioned or implied the use of metrics. Follow-up conversations suggest that students have not internalized the value of metrics and do not yet spontaneously recognize opportunities for their use.

2. How are you documenting your decisions?

Specific technologies (wikis, Google Wave)	8
Engineering notebook	3
Task tickets	2
Requirements document	1
Meeting minutes	1

Contrary to our intention in asking this question, the majority of students responded in terms of specific technologies, such as wikis and Google Wave. None mentioned story cards, any type of formal design documents, code documentation, et cetera.

3. How do you know that you are delivering a quality product?

Client feedback	6
Hard to gauge quality	3
Test-driven development	1
User acceptance testing	1
"Manual testing"	1

Since the nature of our practicum projects is usually to produce a prototype for an industrial client, it is somewhat difficult to define quality in this context. The faculty believe that client satisfaction may be the best overall measure. That said, more granular measures involving usability (if applicable) and code quality (as measured during test-driven development, software inspections, et cetera) should also be employed. Despite emphasis in the formal curriculum, most teams simply did not think of these techniques in the practicum context, or they rejected them given time pressure and heavy client emphasis on demonstrable results.

4. How do you know you are meeting the client's needs?

Discussion with client	9
Formal requirements process	3
User acceptance testing	2
Scrum techniques	1
Addressed needs client had overlooked	1

Given the agile and exploratory nature of nearly all projects, client feedback is, again, probably the best measure of meeting the client's needs. That said, the faculty would like to see evidence of a implementation of a formal process for eliciting, analyzing, and responding to feedback. (Note the inconsistency in that only one respondent mentioned user acceptance testing in answering the previous question and that teams have a minimum of three members.) Our curriculum currently does not provide adequate practice in this area; in the past, faculty members role-played clients during projects in the formal curriculum, but enrollment pressures have led largely to the elimination of this time-consuming practice.

5. How do you know how effective you are as a team?

Internal measures (happy team members, good team spirit, trust)	7
External measures	7
"Not clear"	1

The faculty's preferred answer would involve a combination of external (team effectiveness) and internal (team satisfaction) measures. We would also have liked to have heard that each individual can track both individual and team progress with a positive impact on team spirit. Only one respondent mentioned combining both internal and external measures.

6. How do you run effective meetings?

Agenda	10
"Mix of techniques"	2
Different meetings for different purposes	1
Meetings were not effective	1

The faculty's preferred answer would involve multiple factors, including always having a meeting agenda, having different types of meetings for different purposes, and always having a facilitator. No respondents included all of the factors expected by the faculty, which was somewhat surprising given heavy emphasis on effective meetings throughout the curriculum.

4.2. The use of specific techniques

In this section of the survey, respondents were asked to report on the use of seven specific techniques that had been emphasized in the formal curriculum.

	Yes	"Sort of"	No	Not applicable
Test-driven or test-first development	2	4	8	
Software inspections	5	4	5	
Velocity and/or burndown charts	5	3	6	
Story cards or use cases	9	4	0	
Regular client meetings	14	0	0	
Client feedback on demos	14	0	0	
User testing	2	5	5	2

Given the substantial, often repeated emphasis that these techniques received during the formal curriculum, we expected to see near universal use during the practicum projects. Not surprisingly, all respondents reported meeting regularly with their clients and soliciting client feedback on their work; many commented that this worked well. Surprisingly, the majority of students did not employ quality assurance techniques such as test-driven development or test-first programming, software inspections, and user testing. Also surprisingly, although all students reported using story cards or use cases, two commented that their team's implementation of the technique was poor, and one commented that it did not work well. Finally, it is interesting to note, given that the minimum size of a practicum team was three, the members of a team apparently disagreed on the applicability of user testing to their project.

4. 3. Notable project achievements and omissions

The final section of the survey asked two open-ended questions to allow respondents to comment on their most notable achievements and omissions. Nine responded to the first question; notable, self-reported achievements included:

- Effective work with a geographically dispersed team
- Rapidly learning new technologies
- Rapid, iterative prototyping
- Designing and building custom hardware
- Working closely with the customer.

Ten responded to the second question; notable, self-reported omissions included:

- Software testing
- Risk tracking
- Quality of planning and task tracking
- Bug tracking
- Failure to produce an early mock-up for customer feedback
- Poor user acceptance test criteria.

4.4. Discussion of the survey

We were most struck by the very heavy (perhaps over-) reliance on customer guidance for all aspects of their projects. Virtually no students reported implementing formal acceptance criteria for their work. The 50% weight assigned to client assessment at the time of this survey

(see the changes reported below) may well be a root cause of this observed phenomenon in that it distracted the students' focus from sound software engineering practices to satisfying the client "locally" at all costs. Given that our clients often have little or no experience with software engineering principles and procedures, the client usually does not appreciate work undertaken by the student team early on that has little or no immediately visible relevance to delivering a running system. A related possibility, is that our teaching moved students from the naïve view that agile development is coding without documentation to a somewhat better, but still flawed, view that agile means relying on customer guidance for everything.

Also related to this, the teams tended to rely on customer feedback as the predominant mechanism for quality assurance. Many teams did not implement granular techniques for assessing quality: nearly 60% used neither test-driven development nor test-first programming, 36% did not perform software inspections, and 42% did not do even informal usability testing.

We got conflicting information regarding the use of metrics. In the most positive interpretation, 43% didn't make any use of metrics, and nearly none recognized that the use of metrics provided a good answer to the question "How do you know if your practicum project is on track?" Again, nearly all of those who did use metrics began only after their faculty advisor asked to see them.

Finally, despite a heavy emphasis on effective teamwork running throughout the curriculum, only one student reported that his team used a combination of external and internal measures in assessing team effectiveness. Given that students work in teams, it is puzzling to note that his teammates did not answer similarly.

5. General discussion and future work

When we take a step back and look at outcomes in the large, the situation is not bad. Clients are universally happy with their projects, resulting in a large number of repeat customers, and students feel that the practicum experience provides a valuable capstone to their studies. That said, we have uncovered the need for further study and opportunities for improvement.

A single experience with agile development, even one as in-depth as our Foundations course, may not be sufficient to equip students to *spontaneously* transfer all relevant knowledge and skills to a new situation, especially given the intervening time -- three semesters for part-time students and one for full time. To use Perkins's and Salomon's terms: We are implicitly expecting "low-road transfer" (spontaneous application of well-practiced skills) when perhaps we should be aiding the student in "high-road transfer" (deliberate, effortful abstraction and a search for connections). Our students' emerging skills may not be sufficiently *automatized* or the potential *triggers* in the new situation may not be perceived as sufficiently matching those acquired during initial learning. Also, students may not feel they have the time or license to perform the exploration that might result in high-road transfer [1].

As a test qua possible solution, we have begun providing a framework to scaffold "high-road" transfer, to aid the students in abstracting away project details to see commonalities with previous experience more clearly (cf., "bridging" in [4]). This semester, students answered the first six questions discussed in this paper, as teams, before beginning work on their practicum projects. Their answers were discussed with the faculty advisor and revised, as necessary, if the

answers were inadequate. Student teams will answer the questions again at the conclusion of their projects and will also be asked to comment on how well their chosen techniques worked. (The similar use of such a framework is a hallmark of the proposal-based MSE studio project at Carnegie Mellon's Pittsburgh campus [8]. See [9] for a more comprehensive comparison of the two programs.) Their responses, plus a range of project management considerations, will now count for 30% of their grades, and the contribution of client feedback has been reduced from 50% to 30%. We will also be piloting a new course, *The Craft of Software Development*, which will have a strong emphasis on methods for ensuring software quality.

Finally, we will take a broader look "upstream" at aspects of our teaching in the context of the formal software engineering curriculum and emphasize activities that are likely to promote transfer, most notably (from [1]) providing opportunities for more thorough and diverse practice, encouraging explicit abstraction from specific learning experiences and also encouraging self-monitoring during problem solving.

6. Acknowledgments

We would like to thank the software engineering faculty for their help in composing the survey, our students for interrupting their busy schedules to complete it, and the reviewers for their thoughtful comments on our submitted draft.

7. References

[1] D. N. Perkins & G. Salomon (1992). Transfer of learning. International Encyclopedia of Education (2nd ed.). Oxford, UK: Pergamon Press (accessed via http://learnweb.harvard.edu/alps/thinking/docs/traencyn.htm).

[2] J. Bransford and D. Schwartz, "Rethinking Transfer: A Simple Proposal with Multiple Implications," *Review of Research in Education*, 24(1): 61-100, 1999.

[3] D. K. Detterman & R. J. Sternberg (eds), *Transfer on trial: Intelligence, Cognition, and Instruction*, Ablex Publishing Corp., Norwood, NJ, 1993.

[4[D. N. Perkins & G. Salomon 1988 Teaching for transfer. Educational Leadership 46 (1): 22-32.

[5] R. Bareiss and T. Sedano, "Developing Software Engineering Leaders," Proceedings of the First International Symposium on Tangible Software Engineering Education (STANS-09) Tokyo, Japan, October 2009.

[6] R. Bareiss and M. Radley, "Coaching Via Cognitive Apprenticeship," Proceedings of SIGCSE 2010, Milwaukee, WI, March 2010.

[7] E. Katz, "Software Engineering Practicum Course Experience," Proceedings of CSEET 2010, pp.169-172, 2010.

[8] D. Root, M. Rosso-Llopart, & G. Taran, "Proposal Based Studio Projects: How to Avoid Producing 'Cookie Cutter' Software Engineers," Proceedings of CSEET 2008, pp. 145-151, 2008.

[9] R. Bareiss and M. Rosso-Llopart, "Software Engineering Education at Carnegie Mellon University: One University; Programs Taught in Two Places," *Journal of Systemics, Cybernetics and Informatics*, 7(5): 72-77, 2009.