Form Follows Function: A Trial in Matching the Senior Engineering Course Environment to the Workplace
Steve Chenoweth, Rose-Hulman Institute of Technology
Victoria Bowman, Rose-Hulman Institute of Technology
Sándor Pethes, Rose-Hulman Ventures
Jonathan Labayo, Rose-Hulman Ventures

Abstract
How should senior engineering courses feel to undergraduate students? These courses are the last thing students take before moving into a workplace environment. One logical target is that the courses should feel like that workplace. After all, colleges do something analogous on the front end of engineering students’ experience: The college environment carefully transitions them from high school. At present, however, graduating seniors face workplace environments often much different from their college classrooms, where the pedagogy remains essentially in the format of lectures and homework, followed by a well defined team project. Instructors play a much stronger and crisper role in the outcomes than managers or mentors would in industry, and the coursework is set up so that good students exerting expected effort will achieve a specified level of success.

What would be involved in taking an existing senior engineering course, and trying to match its students’ future workplace environment, within that course? This paper describes one trial in converting such a course, from a standard lecture-and-project format, to a format including more specific elements of industrial workplaces. Those elements included teaching by instructors tied to industry, use of real projects involving system maintenance activities, hand-offs to other teams, close adherence to industry processes like risk management, increased progress reporting, a reduced amount of lecture, and instructors playing management and advisory roles to the teams.

Overall, outcomes were very encouraging, for the idea of employing an environment which resembles industry, in a senior-level course. New problems were introduced, such as student teams getting more unexpected results from their more realistic projects, versus pedagogically well-defined projects. Specifically, students were unaccustomed to achieving less than a high level of project success. Yet the students did learn to deal with realistic problems via their own teams, and using accepted processes. The results suggest that a senior-to-industry “impedance match” generally deserves increased attention in upper division courses. Specifically, they suggest that the pedagogy common to a school can be bent as students approach graduation, in favor of a learning environment more resembling industrial experience.

Key Words
Industrial Leadership, Education Methods, Engineering Curricula, Industrial Partnerships, Innovative Teaching Methods
Introduction

How should senior engineering courses feel to the undergraduate students? These courses are the last thing the student takes before moving into the workplace. Thus, a logical target is that they feel like the workplace environment. On the front end of their college experience, engineering students arrive from high school to a college environment which carefully transitions them: Freshman courses feel similar to high school, though harder and with growing personal accountability. Entering students do better if their high school environment was close to this new one. On the other end, however, graduating seniors face workplace environments often much different from their college classroom environments. Fixing that impedance mismatch, of upper division courses in general versus the next event for a senior, is the subject of this exploration.

Engineering departments strive to fix any content gaps using advice from their industrial boards. And a senior design “capstone” program typically attempts to make the environmental bridge for students. Students get real-world engineering experience in co-ops, internships, and summer jobs, though this experience has wide variability. The question arises, “Why shouldn’t all the other senior engineering courses attempt to match students’ future workplace environment?” Such exposure would provide a wide base of controlled experiential learning.

This paper describes one trial in converting a standard senior-level lecture-and-project software engineering course so that it more closely matches the industrial workplaces into which its students will go.

The course, “Software construction and evolution,” is the final course in a sequence of seven upper-division required courses for software engineering majors. Students were typically beginning their senior year. As a final core course in that program, excepting only the capstone, it is a logical target for making it look close to an industry experience.

We describe prior work supporting our direction, our method, and our results and conclusions. Ingredients were changed in this trial from the prior offerings of the same course so as to give the course more of an industrial atmosphere for students.

Overall, outcomes were very encouraging, for the idea of employing an environment which resembles industry in a senior-level course. These results suggest that a senior-to-industry “impedance match” generally deserves increased attention in upper division courses.
Background on the problem

There is general agreement in computer science (CS) and software engineering (SE) that the curriculum must be designed to achieve specific outcomes needed by people working in this industry\(^1\). Software development is a new discipline with a “growing demand for better, less defect-ridden products.” This has created pressure to incorporate professional practice into the CS/SE curriculum. “Haphazard software engineering techniques are widely recognized as a significant factor in producing software with a high number of defects”\(^1\).

Thus, “graduates of a computer science program must understand not only the theoretical underpinnings of the discipline but also how that theory influences practice”\(^1\). The practical application, of a combination of analytical computing and “soft” skills, is key to a CS/SE student’s future\(^1\). The “soft” skills, such as leadership, teamwork, engineering processes, and professional ethics, are especially difficult to convey to undergraduate engineering students lacking in first-hand knowledge of their value. “…Students enter school without a complete knowledge or appreciation for [professional] issues, which is a source of frustration both for those who teach these students and for those who hire them\(^1\).

On the other hand, there is substantial gain from making the effort to teach these topics to undergraduates. “As students learn more about professional practice and the underlying issues, they become more interested in their studies and [in] how they can work well with others. Incorporating professional practice into the curriculum can therefore serve as a catalyst to awaken and broaden a student’s interest in computing”\(^1\).

Possible solutions

The traditional solution to this tricky need for soft knowledge has been to let the workplace provide it. However, this produces a lag in the college graduate’s productivity. “Most recent college graduates require a substantial amount of on-the-job training before becoming productive because most of them have not had the opportunity to develop commercial software while in school”\(^2\). The process and social skills are becoming increasingly important to employers\(^3\). At the same time, companies also have become much more demanding as to the level of industrial maturity expected of graduating seniors.

Another option is for students to take advantage of industrial co-ops and internships and so “mature faster in their problem-solving skills and become more serious about their education”\(^1\). However, combined, “sandwich” programs integrating co-op experience most often extend the time until graduation by at least a year.

An alternative to parallel training and education is to bring these two worlds closer together. For example, “One of the most important ways that the private and public sectors can support the education process is to encourage their employees to play a greater role in helping to train students. These employees can offer support in a number of ways” such as functioning as mentors of students working on projects\(^1\).
The engineering capstone was supposed to play a major role in closing the impedance mismatch between school and industry. Yet, how well does that alone serve the purpose, and why would one part of the engineering program be singled out for this role, versus, say, other upper-division courses in an engineering major? 

It is true that “many academics would argue that this is really ‘training’ rather than ‘education’ and the rightful place for such preparation is at the companies’ own workplaces. The counter argument is that as the lessons learned seem to be so significant, any course that does not include some element of real world teaching is failing to give the complete picture”.

Additionally, “Without prior exposure to the expectations of industrial demands, students have a hard time adjusting to jobs in the software industry after graduation”. In one study, “it was not just a lack of knowledge of the company tools and product line that reduced the graduates’ effectiveness in the early months, but rather it was their lack of awareness of and preparation for the realities of the workplace such as changing objectives, problems with clients, or the pressures of imposed deadlines”.

Testing for deep knowledge of professional practices is not easy, because it is essentially heuristics and judgment calls, and these differ depending on the situation. Some parts of this “soft knowledge” problem may be that, in software engineering at least, there is more reliance on experience versus principles because it’s a less mature field than other engineering fields. It is not clear that this is the whole problem, however. Software also does not inherently rely on physical laws as much as other types of engineering, for example. And, because it is portable, there may be more novelty in each development project.

Communicating industrial reality more effectively is one of the remaining problems of software engineering education. Many challenges are encountered with teaching such "practical" courses. For example, “The learning curve for fulfilling software professional roles varies among students. Some students adapt to their roles quickly, while other will take a longer time to get used to their assigned roles”. Generally, “It is difficult to quote anything other than informal feedback to justify the claim that these courses are very successful”.

**Making projects realistic**

A very common practice to deal with the impedance mismatch is to try to make course projects as realistic as possible. Making class projects “real” is not easy, however – there can be issues even with graduate students who have experience in industry. A significant step toward this end is to have real customers, although an issue in this case can be the actual value to that customer, of the project they propose for class; this issue is often encountered in capstone projects. The underlying concern here is that students lend value to projects based on knowing they have real value.

An additional, critical issue is that these projects must be large and with enough complexity to make process and other professional practices clearly important. The size of the program being worked on is important in showing process lessons. That problem has been cited as generally
embedded in CS/SE curricula\textsuperscript{19}. To overcome this, some schools have students across multiple classes participate in building a large system within the department\textsuperscript{20}.

Students also need to feel they have had experience at performing realistic professional roles. As an example of a success story here – “A two-course sequence in System Analysis and Design provides students with the opportunity to experience ‘real world’ situations through team-based projects. Students assume various roles as professionals do in the software industry and are expected to contribute to their team and project”\textsuperscript{7}.

The role gap also can be closed by showing real people from industry in action or otherwise finding ways to include learning from industry experts\textsuperscript{2}. In terms of teaching style, the traditional classroom experience, with its lectures, exams, and homework, has been marked as a major influence which should be reduced for students to become ready for industry. Instructors need to evolve into looser roles like monitor/observer, commenting on the project work. Students need opportunities to reflect and reach their own conclusions\textsuperscript{5}. Businesses need workers who can think about multiple issues at once, which is not what students are trained for in instructor-led classroom courses\textsuperscript{21}. Students need to learn decision making amid chaos\textsuperscript{22}.

As an example of a way to simulate real software projects, Dawson, \textit{et al.} \textsuperscript{6} introduced “dirty tricks” in student projects, with informal means of deciding on success. Their tricks included:

- Give an inadequate specification
- Make sure all assumptions are wrong
- Change the requirements and priorities
- Have conflicting requirements and pressures
- Give additional tasks to disrupt the schedule
- Introduce quality inspections
- Change the team
- Change the hardware

We list these examples of things these authors pursued in class because, if they weren’t done to prepare students for industry, many of these tricks would be considered bad pedagogical practices. The conflict in practices highlights the need for instructors to be familiar with the industrial environment.

\textit{The potential role of open source software}

A growing way used to include realistic projects in courses is to make these projects “open source”\textsuperscript{13,23}. Open source software is an active area of industrial development\textsuperscript{24}. Tools like Eclipse and operating systems like Linux are prime examples. Many of these are now used as a routine part of the environment in CS / SE classes, as well as in industry. However, beyond just use of such software, students can engage in adding to the software projects at sites like SourceForge\textsuperscript{25}. Many of the systems are much larger than what a student could create on their own. Students simultaneously can see how and with what processes real software is built. Open
source projects lend themselves to several industry practices. And students can judge first-hand about the merits and issues of open source software by trying to contribute to it.

Kal Toth and others have used open source projects for senior project work. These projects also are valuable for courses in software evolution (i.e., maintenance) because they provide a starting point for the addition of new features. This is the core of the method we used in the course described here.

**Study Method**

We have added more realistic, open source projects and other industry-like features to one course in particular. This paper reports on our study of that course. The course in fact has been enhanced over multiple years, with the goal, of the present authors and others, being to discover how much industry practice can be included in it, and how much the non-industry aspects can be reduced. We focus on the impacts of the most recent changes, while also noting earlier changes and their effects, which were documented at the time in course assessment reports.

The course, “Software construction and evolution,” is the final course in a sequence of seven upper-division requirements for software engineering majors. Students were typically beginning their senior year and simultaneously starting their capstone experience. The course content is to learn processes, heuristics, and best practices related to implementation of software. This is the coding – the building step of software development. The students all are experienced coders, so there is an expectation in the course that they are very good at this activity already. Also included are detailed design practices and testing practices surrounding that coding. Finally, students are introduced to issues surrounding software maintenance, which is in fact the activity done most of the time in many software development organizations – enhancing and fixing existing software, often for some new customer or for some increased purpose for an existing customer or customer base. An example of “soft principles” from this area is the “laws of software maintenance,” such as the following, “As a program is evolved its complexity increases unless work is done to maintain or reduce it.”

**Course history**

Prior to the 2005-6 school year, the course was more traditional in design, with projects taken from other classes or provided by the instructor. The course had a moderate workload, including homework assignments, four exams, and regular project progress reporting and other process emphasis – on not exceptionally difficult projects. Students in the class suggested possibly finding more challenging projects, such as larger ones using the C++ language from industry. They asked for more class time for team meetings, fewer exams, and smaller team sizes with more projects. They also suggested that code tuning be more emphasized – a skill building on their prior expertise at coding.

During the 2005-6 school year, open source projects in C++ were tried, with the emphasis shifting to harder maintenance projects. Contributing back to the open source projects was included as a project option. For the 2006-7 school year, these open-source projects then became a strong recommendation from the students. It was at this time also that the course was moved
from the junior year to the senior year in the required course schedule, making it the logical culmination of prior software engineering courses.

After these years of incremental course changes, a more radical matching to industry was considered for the 2007-8 school year. This was partly because of the course’s new position in the curriculum, but also to see first hand how far that industry matching could be taken for a non-capstone course. Key ingredients changed in this trial, from the prior offerings of the course, were as follows:

- Use of multiple instructors playing advisory roles such as product / project manager and technical expert.
- Instructors all closely tied to industry. One was on loan from an industrial partner (Tori Bowman); two others worked as liaisons from industry (Jon Labayo and Sándor Pethes).
- Real projects instead of fictional or “inside” projects.
- Shorter projects, resembling most short-term product maintenance work.
- Project hand-offs.
- Teams picked for expertise by “management” to match project needs.
- Close adherence to a specified process used in industry.
- Frequent milestones and weekly reports by teams.
- Less lecture and other traditional class ingredients.
- More project time in class, with issues handled there.

Some pertinent details of and connections among these teaching methods are discussed in the next paragraphs.

Students were organized into teams of three and asked to track their roles and effort on two short maintenance projects. Each team also had two instructors acting as project / product managers, and two instructors acting as technical experts. The instructors all had recent or current backgrounds working in these areas in industry. As an organizing activity, instructors picked members for each team to match project needs. For example, each team had at least one experienced C++ programmer and two students who had expressed interest in the particular project.

All of the projects were required to be from open source sites: real projects for which maintenance would be meaningful to an external audience. The open source projects were picked by students through a triage process in which all students initially suggested one, then signed up for each other’s projects.

There were two 3-week projects, resembling short-term product maintenance work. In between these projects, the teams handed what they had done to another team in the class. Thus, they had to document their work-in-progress for a second team to take over. This is an example of an industry-based required process. As another example of industry process, in the second project students had to assess and track risks carefully and work from the basis of reducing those risks. The frequent milestones and weekly reports by teams also reflected current industry practice on maintenance projects.
The short projects involved other industry-like processes and steps: development of a project plan for the maintenance, the implementation of that plan, development of a software manual, delivery of the maintained code and supporting documents including test cases and results, development of a transition plan, a peer evaluation among project team members, a supervisor’s evaluation of the project team, and a final oral presentation.

Lecture in the course already had been reduced to half of the class meetings in the 2006-7 version of the course, and this was retained. Three of the weekly homework assignments were eliminated. The two tests done the previous time (versus four originally) were retained. The project time in class included designated days when the technical experts would be available for face-to-face help.

**Assessment tools**

To assess the success of these changes, anonymous student evaluations were used, along with feedback from the four instructors and assessment of student work. Many of these materials also were available for comparison from earlier classes, including student work from prior years.

The maintenance projects provided artifacts against which course objectives could be measured. The students built a maintenance project plan and a test plan, the maintained code itself, and an implementation / transition plan. There was a peer evaluation among project team members and also the supervisor’s evaluation of the project team. Oral weekly progress reports were given, and a final oral presentation included reflection on cause and effect.

Leading up to the expertise expected on the projects, students did one homework per week (almost all individual work), where they applied material from the course lectures and discussions. Each lecture also had a daily quiz used for immediate feedback. The two exams were used to corroborate individual student knowledge and capabilities regarding the course material. They included some program development in C++.

**Findings**

Use of real projects especially produced unexpected events and drove additional as-needed learning into the course. The students all achieved the predefined learning objectives. Each team met the process outcomes. Their projects were not all successful, but because students ran into many unexpected twists and did deal well with many of those, their learning was high. Full project success was not required to meet course objectives.

Overall, outcomes were very encouraging for the idea of employing an environment which resembles industry in a senior-level course. The following sections describe how the running of this course, and its outcomes, were seen in retrospect by the four instructors. Of our four authors, Jon and Sándor were the technical experts, Tori was a visiting professor on loan from the software industry, and Steve was a regular professor with 30 years of experience in the industry. See their bios for more information.

**Results from Jon’s and Sándor’s perspectives**
Working as project managers and mentors at Rose-Hulman Ventures, integrating with the project work in this course felt very natural.

While project liaisons were available for technical expertise in class each week, we found most of the interaction occurred outside the scheduled class time. The amount of involvement varied based on the familiarity each team had with their target technology, and increased when the teams faced challenges. Most of the interaction took place over email, frequently occurring in the evenings and weekends. Although we were involved in an advisory capacity opposed to representing the client’s needs, being located at a satellite campus required the teams to take a more active role in seeking out assistance; teams would need to plan out in advanced scheduled times to meet with us and be prepared.

Removing the restriction that projects involve a specific language would be a welcome evolution of this experiment. While we personally feel that the nuances of managed languages such as C++ are important for students to learn, it became a distraction in this course and the material would fit better elsewhere in the curriculum. Giving the teams more flexibility in choosing a development environment they were more comfortable with would have minimized this distraction, but special consideration for this may need to be taken into account when the projects were to handed off to the alternate teams.

Compared to the CS/SE capstone project, the work in this class focused more on risk assessment, execution and maintenance of a project than specifications. Students were able to experience the tribulations of working with legacy code and risk identification/mitigation directly. Unlike the capstone project, each team represented the client and did not have to solicit requirements of specifications.

The project hand-off presented an interesting learning experience for the students. Due to the type of work that we do, we are often primarily concerned with the work output for the project; in this case, we felt that the educational value of the risk assessment, documentation, maintenance, etc. outweighed the necessity of successful completion of implementation.

**Results from Tori’s perspective**

As a project manager, one major deficit I see in new-hires is a lack of foresight in software development. Generally speaking, most new software engineers don’t start with a formalized plan or risk assessment. They dive in head-long and just hope for the best. My primary objective for this course was to help the current seniors realize the repercussions of this – both in their own hands on experiences and through ‘war stories’ of my own.

I put a major emphasis on risk assessment and monitoring. As mentioned previously, this was made a requirement during the second project phase. At the closing of the first half of the project work, almost all of the groups made mention of the gross underestimates – both in terms of the time needed and skills needed on the team. This reflection provided a strong basis for the risk analysis.
As a part of the brief weekly standup status reports required by the teams (something I employ with my teams as well), they were to status their risks – both in the presentation itself and the written status update. This included not only the risk itself, but also the impact, the mitigation strategy, and key resources involved. As should be a part of any good risk analysis, the teams were to identify one opportunity they were going to attempt to capitalize on.

Examples of common risks identified included inability to recreate the bug reported on the team’s systems, conflicting priorities among team members, and integration of new features into the existing code base. Most of the opportunities indentified centered on reducing the time required or the ability to add additional ‘wants’ in the same timeframe. Examples of common opportunities identified included finding exceptional reference material and decrease in team commitments to other projects. All of these are staples to industry risk analyses.

During the successive status reports, it was evident that the plans were being adjusted to reduce the probability and/or impact of the risks identified, or to increase the probability and/or impact of the opportunity identified. The final presentation for the second half found most of the teams had been able to reduced or eliminate one or more of the indentified risks.

In my opinion, this will be an invaluable experience for each of the students as they enter the workforce. In some of the student evaluations, the students lamented about the amount of upfront planning we required from them for such a small project. Even though some of the students didn’t appreciate this particular aspect of their learning experience, they will be much more acclimated to software engineering in industry.

Results from Steve’s perspective

We had some success with our course changes, but also ran into some new issues.

The students’ recommendation from earlier classes to cut back on regular class work to allow project time may have backfired. Some students specifically noted that they only got 20 lectures out of 40 classes, and therefore, felt shortchanged. This made the course feel especially light in terms of theory.

We allowed students to pick more challenging projects, also responding to student comments from a prior course, ruling out only the most ambitious (like “rewrite an application”). Their optimism about what they could achieve caused most of the teams to be overly ambitious.

Students reacted in highly varied ways to their projects succeeding or not in the same ways as similar industrial maintenance projects: some of the students already appreciated our introducing this reality; one student saw us as demeaning; one took out his/her frustration on open source software; one concluded s/he had “failed to grasp any new concepts.”

Some students did not like having to do open source projects or projects in C++ and would have preferred to have a choice. This may have been exacerbated by having instructors available only part of the time as the support for this environment.
Conclusions

These results suggest that a senior-to-industry “impedance match” deserves increased attention in upper division courses generally. For an engineering department, this course delivery environment would go beyond just providing content aligned with industrial recommendations, yet all taught with standard pedagogical methods. It also would move beyond what is typically in current practice in colleges – having the only sample of standardized industrial experience be provided by the capstone, the senior design program.

The idea is now widespread, that the content of senior-level engineering courses should close to that of industry. The course studied here exemplifies that trend. The accompanying question is, to what extent should the pedagogy common to a school also give way, in favor of learning in an environment more resembling industrial experience? How much should the “form follow function”30 in courses which do transition a student to industry?

Even as seniors, most of our students are still accustomed to standard pedagogy. Perhaps any first course they encounter which changes this pedagogy will receive varied feedback. Students also are accustomed to having almost everything they do in class succeed almost entirely, which will not be their destiny in the software industry. A strategic management decision may be needed, as to whether even a course like this – the last in a required SE majors’ sequence – can and should be taught in a manner close to industry standards.

In extensions to this study, one possible addition would be to add variations in teaming situations. For example, some teams could be allowed good start-up times with few if any rushed tasks, while other teams are pushed to produce. This would lead to different stress levels within the teams.

Bibliography

6. Dawson, R. “Twenty dirty tricks to train software engineers.” Proceedings of the 22nd international conference on Software engineering (ICSE) ’00.


10. Freeman, P. “Realism, style, and design: Packing it into a constrained course.” Proceedings of the ACM SIGCSE-SIGCUE technical symposium on Computer science and education, Volume 8, 10 Issue 1, SI, February 1976.


Steve Chenoweth is an Associate Professor of Computer Science and Software Engineering at Rose-Hulman Institute of Technology. Prior to this, Steve worked at Bell Laboratories, where he conducted systems architecture reviews, and at NCR Corporation, where he designed and built transaction processing systems. In both of these long-term, prior jobs he taught software engineering in industry at the same time. Email: chenowet@rose-hulman.edu.

Tori Bowman is a Visiting Professor of Software Engineering at Rose-Hulman Institute of Technology. Prior to this, Tori was a systems engineer and project manager at Rockwell Collins in Cedar Rapids, Iowa. Email: bowmanvc@rose-hulman.edu.

Sándor Pethes is a senior project manager for Rose-Hulman Ventures, a department of Rose-Hulman Institute of Technology which provides professional experience opportunities for students. Sándor’s primary focus is managing student teams that execute projects for industry clients. He has a background in Computer Science. Email: pethessa@rose-hulman.edu.

Jon Labayo is a senior project manager for Rose-Hulman Ventures. He is primarily involved with managing student teams doing a variety of different projects, ranging from laboratory control systems to prototype consumer products. His background is in Computer Engineering. Email: labayojb@rose-hulman.edu.