Int. J. Engng Ed. Vol. 00, No. 0, pp. 1–12, 2008
Printed in Great Britain.

0949-149X/91 $3.00+0.00

The Software Enterprise: Practicing Best Practices in Software Engineering Education*

KEVIN A. GARY
Division of Computing Studies, Arizona State University at the Polytechnic Campus, Mesa, AZ 85212, USA. E-mail: kgary@asu.edu

Software engineering educators emphasize teaching concepts in software engineering principles and then applying them in the context of a capstone project. Capstone experiences often focus on leveraging a popular process model. The emphasis on process provides a structure for coordinating team activity, with an objective of demonstrating to the student the value of following a process model. We contend that more emphasis is required on detailed process execution than is given proper due. Specifically, best practices are now emphasized in the software engineering profession over rigid process structures, and as educators we must respond to this cultural shift by teaching the role of best practices in a broader applied process context. Our approach in the Software Enterprise, our multi-year capstone sequence at Arizona State University Polytechnic, is to provide a process structure, teach best practices, and then give teams 'just enough rope' to resolve issues by leveraging the process, best practices, and soft skills. The Software Enterprise presents a unique, iterative accelerator for presenting software engineering from concepts through to applied practice. This pedagogical model allows us to present, practice, and apply best practices in the context of real scalable projects, resulting in better contextual learning for our students. In this paper we describe the machinery for teaching software engineering in this manner and present some preliminary survey results evaluating how well Enterprise students apply these skills in practice.

Keywords: engineering education; software process; software engineering capstone; software enterprise

INTRODUCTION

SOFTWARE ENGINEERING has a high impact on the national economy. A report from the ITAA maintains that computer programming remains the largest segment of the information technology (IT) workforce, representing 20 percent, or over 2 million jobs†. ITAA Projections for future employment growth are strong as well, with programming ranked third out of nine IT categories‡. The Bureau of Labor and Statistics forecasts 45 percent growth in Software Engineering opportunities from 2002 to 2012, representing over 310 thousand jobs. Software Engineering for applications ranks 8th and Software Engineering for Systems Software ranks 9th out of *all* career categories§. Salary.com/Money magazine's recent survey¶ ranked Software Engineering the #1 career field. These numbers are important, as they indicate that despite perceptions regarding outsourcing and the decline of computing enrollees, economic demand for competent software developers and software engineers is in fact increasing.

To respond to this need, the Division of Computing Studies (DCST) on the Arizona State University Polytechnic campus is tasked with producing 'industry-ready' graduates. In the model of a polytechnic, DCST places an emphasis on hands-on practice over pure scientific study. Through this emphasis, input from industrial advisors, and lessons learned from DCST's previous one semester capstone project course, the capstone experience underwent a significant evolution. The scale and scope of the capstone experience was extended to a multi-semester, multi-project, and multi-year sequence starting the fall of 2004.

Graduates should exhibit a higher degree of applied competencies in industry-relevant areas, and as an indirect measure should have more success in career placement and advancement. To achieve applied competency in industry-relevant areas, we focus on process in the capstone project. We use the Rational Unified Process ('RUP' [19]) as the encompassing process model, but borrow heavily from Agile methods for *best practices* in use in industry. Example best practices include an emphasis on unit testing and test-driven develop-

* Accepted 25 April 2008.

† Information Technology Association of America. *Adding Value . . . Growing Careers: The Employment Outlook in Today's Increasingly Competitive IT Job Market*. Annual Workforce Development Survey, September 2004. Fig. 3, p. 8.

‡ Ibid. Table 20, p. 22.

§ US Dept. of Labor, Bureau of Labor Statistics. 'Tomorrow's Jobs', reprint section from the *Occupational Outlook Handbook*, 2004–2005.

¶ Money magazine/Salary.com *Best Jobs in America*, April 24, 2006. http://money.cnn.com. This survey was also based on BLS data, but weighted in areas related to compensation and job growth (see http://money.cnn.com/2006/04/10/pf/bestjobs_howwepicked/index.htm

ment, evaluation of third-party (COTS and OSS) dependencies, configuration management patterns, refactoring, and continuous builds and integration testing. These practices are presented in a unique pedagogical model geared toward accelerating a novice from initial concept to applied competency. Students are presented with the motivational concepts for the best practice, apply the concept via state-of-practice tools in a lab setting, and then apply the concept in a scalable project context. The result, we intend, is a student better prepared to be immediately productive in the workforce.

This paper presents the Software Enterprise pedagogical model, and describes the best practices emphasized within the process context. To gain a better understanding as to whether this model is accelerating the industry-preparedness of our students, we conducted a survey of current students and recent graduates. We share the results of this survey and our interpretation. We compare our experience with similar approaches in academia, and conclude with some thoughts about the potential impact of this model on software engineering education.

PEDAGOGICAL MODEL

A common challenge in creating software engineering curricula is that most programs start as an outgrowth of an existing computer science program. Therefore, coursework must be integrated in some fashion with a breadth of computing requirements. The ACM/IEEE recommendations [17] suggest several curricular patterns, outlining different ways to introduce software engineering courses and topics into an existing program. This is certainly practical; faculty in this area must find a way to work within existing program structures with a slant toward efficiency in doing so. However, one must wonder if the resulting structures resemble anything like a software engineering program if considered with a clean slate.

Another issue is the recommended curricular structure assumed for teaching software engineering. Table 1 shows the general pattern from the 2004 ACM/IEEE Joint Task Force recommendations.

Table 1 is the recommended pattern in [17] for schools adopting a 'Computer Science first' curricular pattern. A 'Software Engineering first' pattern is also presented, but the main issue is not which is first, but the software engineering pattern within the pattern. In all the recommended curricular structures, Software Engineering is presented first through a breadth-oriented course (or two-course sequence, SE200/201 in Table 1), followed by an unordered collection of courses in individual process phases or skill domains. Typical courses might include 'Requirements Analysis', 'Quality Assurance and Testing', 'Formal Methods', 'Project Management'; the 'SE' blocks in semesters 2B–4A of Table 1. At the conclusion of these courses, a one or two semester capstone course awaits (SE400 of Table 1).

We have several issues with this approach. First, we find that most students do not have any background to grasp the concepts and issues of software engineering presented in the survey-oriented course. The result is students who can recite the classic pros and cons of various process models and techniques, but really have no foundation as to why these axioms are accepted in practice today. They lack the ability to think critically about their emerging profession. Second, the capstone project is seen as a rite-of-passage as opposed to a true learning experience. The lifecycle is compressed and addressed in the wrong order. Specifically, students typically work on requirements, then design, then develop, then test on a clean project. We contend this does not reflect the experience when students graduate. Instead, students should participate on a team in the order their professional careers are likely to evolve—test, develop, elicit, analyze, design, then manage. The Enterprise course sequence defined for the two-year experience follows this pattern, and is presented in the section 'Curricular and process model'.

Third, software engineering education tends to emphasize the *macro level*. By this we mean the process phases of the software lifecycle; requirements, maintenance, quality assurance, etc. Our experience, particularly with small enterprises and open source software, tells us that software engineering is becoming increasingly *agile*, re-focusing on *micro-level* activities that impact on how software is developed on a daily basis. To this end, the Enterprise incorporates industry-oriented best practices taken from Agile methodologies, even without adopting Agile methods at the macro level. This focus is discussed further in the section 'Best practices'.

Table 1. Recommended software engineering curricular pattern ('Pattern SE' from [17], p. 60)

Year 1		Year 2		Year 3		Year 4	
Sem 1A	Sem 1B	Sem 2A	Sem 2B	Sem 3A	Sem 3B	Sem 4A	Sem 4B
Intro Computing Sequence			CS(Int)	CS(Int)	CS(Int)	SE400	SE400
CS105	CS106	*Calc 1*	*Calc 2*	MA271	SE	SE	*Tech elect*
NT		SE200/201	SE	SE	SE	*Tech elect*	*Tech elect*
		NT	SE	NT	*Tech elect*		

Table 2. Student participation trajectory

Term	Fall		Spring	
Course	Title	Focus	Title	Focus
Year 1	Software Tools	PSP, productive use of software tools, work w/ existing software	Construction and Transition	Software development best practices, deployment
Year 2	Inception and Elaboration	Project vision, requirements elicitation, analysis, prototyping	Project and Process Management	Project planning, monitoring, and tracking

Finally, the individual subject areas teach skills in a particular technique, language, or methodology, without the context of the entire lifecycle. The emphasis should not be on the individual areas, but on their connections; on how they work as a whole and in context. The pedagogical delivery mode of the Software Enterprise addresses this concern, and is discussed in the section 'Delivery model'.

Curricular and process model

The Software Enterprise curriculum plan calls for four upper-division courses to be taken as a sequence during the last two years of study. During this time, students participate on teams that complete two one-year projects. This sequence is shown in Table 2.

A significant decision for capstone project facilitators is the software process model to follow. Choices include some that are prescriptive and more easily supported in the classroom, such as the PSP/TSP [14–16] or Waterfall models, while others are iterative and more flexible such as Agile/XP [1], RUP [19], and Spiral/Theory–W models [2, 3].

A student entering the Enterprise sequence (Fall term for Year 1 in Table 2) begins by taking a Software Tools course. In this course a student gains exposure to a set of tools that support the software process. The tools are exposed through the Eclipse IDE*. The student joins a project team, assisting with prototyping for requirements validation. A student's second semester (Spring term for Year 1 in Table 2) is spent in Construction and Transition. Students spend significant time developing the software according to specific project requirements. Students are also responsible for transitioning activities such as packaging, deployment scripts, performance and scalability testing, and product documentation. The completion of this semester also marks the completion of the student's first project.

In the third semester (Fall term for Year 2 in Table 2), a student begins a new project by starting with product Inception and Elaboration phases. Under constraints provided by the course facilitator, students elicit requirements from external project sponsors, and perform requirements analysis resulting in a logical model of the system. The logical model is validated through user interface (UI) and architectural prototypes. The UI prototype is created with a storyboarding tool† (whenever possible) with the purpose of validating flow and usability with customers. The architectural prototype requires teams to experiment with existing software (from previous projects, project sponsor code repositories, or open source) to understand how to integrate these technologies into an architecture to support the full implementation in the Spring. Teams produce a Vision, SRS, Software Development Plan (SDP), and Architecture documents. These documents and the logical model serve as the input product for the next semester's implementation phase.

In the fourth and final semester of the Enterprise sequence, a student serves as a process manager, test planner, and mentor. As process manager, fourth semester students are responsible for process planning, process monitoring, and process changes. Process planning is the selection and instantiation of specific process practices within the iterative RUP framework. For example, teams may choose to do test-driven development (TDD) early in Construction to ensure requirements are understood, but then abandon this practice in a later iteration. These decisions are documented in the team journal. Process monitoring is performed through weekly status reports, in-class 'PMO' meetings, and earned-value analysis. The instructor may introduce random process changes to requirements, schedules, and resources. Teams must handle these events according to the instantiated process model. Fourth semester students are also responsible for writing the test, deployment, and release plans for their software products.

The fourth semester students plan the Construction and Transition activities of the second semester students. This arrangement allows upperclassmen to mentor lower-division students in a highly interactive manner. Co-located weekly lab meetings facilitate this collaborative and mentoring relationship. We also allow first-year graduate students to enroll in the second year (Year 2), which has brought to light cultural norms differ-

* The Fall 2007 semester was the first time that we used IBM's Jazz environment (http://jazz.net), which is built on top of Eclipse.

† Storyboarding tools include iRise (http://www.iRise.com), rapid prototyping languages, and index cards.

Table 3. Software Enterprise course topics by course sequence

Course 1	Course 2	Course 3	Course 4
Intro to PSP	GUI development	Software lifecycle process Review	Software development Planning
Eclipse	Software construction	Requirements engineering	Task Identification/WBS
Ant	Unit testing	Requirements Documentation	PERT and critical path Analysis
Use case diagrams	Test-driven development	Requirements elicitation	Task scheduling/Gantt charts
White/Black box testing	Defensive programming	Use cases	Estimation
System testing	Refactoring	User stories	Risk management
Metric tools	Code reviews	Requirements quality	Inspections
	Metrics	Requirements analysis Overview	Test Planning
	Configuration Management	RUP analysis	Release management
	Professionalism and ethics	Structured analysis	Postmortem
		Usability	
		Requirements management	

ences between American and international students. Mentoring and cultural issues are further discussed in [11].

Best practices

For each course in Table 2, a set of course topics has been identified as part of industry best practices. This list is an amalgamation of traditional software engineering topics and current best practices. Feedback on the topics, listed in Table 3, is provided through DCST industry partners.

The specific course topics are of interest because they represent a cross section of best practices from software development processes. These practices are becoming industry norms, as the community moves toward lightweight methods and away from document-centric models.

In some cases the practice may be mentioned more than once, particularly when considering topics in Course 1. Several of the topics in Course 1 are used to 'set up' later activities in the context of projects. For example, the PSP is covered in about one month, certainly not sufficient time to cover it in the manner prescribed by Humphrey []. The purpose is to create better personal estimation skills so these students can provide better estimates to their Course 4 counterparts that must do project-level estimation activities in the Spring. Likewise, several tools are exposed to students in areas such as unit testing and metrics, while in Course 2 these practices are revisited for conceptual grounding alongside tool knowledge.

One may conjecture that some of these best practices are mere fads that will no longer be relevant in a few years time. This may be the case, and the solution is to simply change the practices. *Best practices* are constantly evolving in the professional community, and it is academia's task to keep up, not the other way around. More durable are the process models in which the practices are exposed. Iterative and incremental process families for some time have been considered the best process models, with no foreseeable change on the horizon.

Delivery model

We emphasize capstone projects as learning vehicles rather than rites of passage. We recognize the need to expose students to proper foundations and to show them how they apply the concepts. We therefore designed an iterative feedback pedagogical model that incorporates traditional dissemination activities with practice and application. This model is shown in Fig. 1.

Figure 1 depicts the pedagogical delivery model for software engineering best practices in the Software Enterprise. Students are first exposed to concepts via traditional dissemination activities; lectures, readings, and discussion. They then put the skill into practice using an industry-accepted tool in a proctored lab exercise. The practice is then adapted and incorporated into the process model in a scalable capstone project. Students complete the cycle by reflecting on the lessons

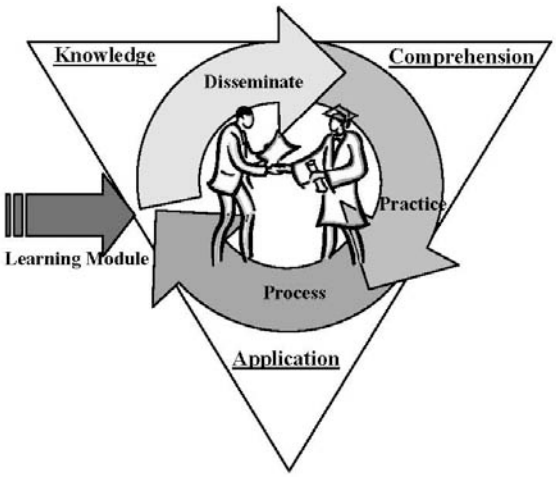

Fig. 1. Software Enterprise Pedagogical Delivery Model

learned while applying the technique on their project. The key is these activities take place closely in time as opposed to over a several semester period.

As an example, consider a concept such as configuration management (CM). CM concepts such as quality thresholds for codelines, good and bad branching patterns, and change management are presented in regular lecture meetings. At the end of the same week, CM is grounded via a half-day laboratory exercise in a tool (formerly CVS, we now use IBM's new Jazz environment). The following week, CM is incorporated into the capstone project. At the end of the current three-week project iteration, teams are required to write in their team journal (a blog or Wiki), a rationale for how they applied the technique into the project and an evaluation of how well the approach worked.

We believe the coupling of disseminated knowledge to skills practice to incorporated process tasks leads to quicker comprehension and deeper applied knowledge than the traditional model. We refer to this model as an 'Iterative Instructor-facilitated, Learner-centered' model. Supporting such a highly iterative teaching and learning methodology places a great burden on instructors-as-facilitators to lead students down the right path. Knowledge from disparate sources must be filtered, aggregated, and packaged for digestion in a practice-oriented, collaborative learning environment. Structured, hands-on exercises for problem-centered learning must be constructed. Facilitators must determine the correct amount of guidance and support to provide team projects that enable learning without causing projects to degenerate into a 'thrashing' state, alienating students from finding the right path. Finally, and most importantly, instructors must rethink how learning is assessed, and how to assess the relative success of the Enterprise sequence.

ASSESSMENT

The Software Enterprise's vision is to produce 'industry-ready' software engineers. In our view, greater applied knowledge means a more 'ready-to-perform' graduate, and as such we should see a greater adaptation to industry. In the model of the Lethbridge study [20], we designed and implemented an assessment methodology for tracking needs and competencies of students versus recent graduates. This objective was accomplished by designing, implementing, and delivering an online survey targeting recent graduates, and comparing the results with data gathered from students while in school. We summarize our data and findings in this section. Please note that we do not have a population size that allows for rigorous statistical analysis; however we believe these early data do provide some useful trends for us to be aware of as we move forward. Furthermore, these surveys results should be interpreted alongside additional assessment data on the perceptions of students in the Enterprise. These data are presented elsewhere in the literature [8, 11] so we do not repeat them in full here, but we do include in Table 4 student perceptions of topics to which they have been exposed, and expectations of professional benefit from [11].

As we are focused on 'industry-readiness', we show only data for prior academic exposure and professional benefit. What we find interesting in this data is that students perceive professional value in software engineering topics yet are not exposed to them until they enter the Enterprise sequence. In fact, the most telling concern in this table is the recognition by our students of the need for these practices to be learned to benefit them professionally, but the disproportionate emphasis in their academic time. We share these results as we believe they are a source for a bias we discover in our survey below.

Table 4. Results of student survey regarding topics to which they were exposed and that have professional benefit

Survey results	Academic exposure			Professional benefit		
Topic area	None	Some	Lot	Lot	Some	None
Code reviews	57%	36%	8%	57%	43%	
Configuration management	91%	9%		91%	9%	
Defensive programming	36%	54%	9%	80%	20%	
IDEs (Eclipse)	15%	54%	31%	82%	9%	9%
Metrics	82%	9%	9%	40%	20%	40%
Refactoring	73%	27%	9%	82%	18%	
Deployment/Release mgmt	92%	9%		67%	33%	
Unit Testing	36%	45%	18%	90%	10%	
Estimation	77%	23%		83%	17%	
Project management	69%	23%	8%	77%	23%	
Quality planning	57%	22%	22%	92%	8%	
Release management	83%	17%		62%	23%	15%
Defect tracking	67%	11%	22%	71%	29%	
Risk management	53%	33%	13%	92%	8%	
Task planning and sequencing	53%	20%	27%	92%	8%	
Test types (alpha, beta)	75%	8%	17%	50%	42%	8%
Analysis modeling	75%	25%		57%	42%	

Study methodology

We wanted to conduct surveys in the model of the Lethbridge study [20] to assess where there might be gaps between student expectations and reality once students graduate and begin working their first job. Table 4 shows student expectations. In this section we describe the results of an online survey effort conducted in December 2006 to assess how our recent graduates fared.

We compiled an email list of all Enterprise students and graduates from Spring 2004 onwards. From this list of 84 names we were able to successfully email 82 people (we cannot confirm they actually received the e-mail, only that they did not bounce). Of these 82, 29 took part in the anonymous survey. The survey asked demographic questions so we could categorize indicators such as the major when in school, how long ago the student graduated, job field, GPA in school, and size of current company. The survey asked respondents to assign need and ability in general and technical competencies on a scale of 1 to 10. The survey questions were as follows:

Question 6: For each of the following *general skill areas*, indicate the level of expertise needed in the area to perform your job successfully. Use the following interpretations for levels 1, 5, and 10:

(1) 'I need to be familiar only with the very basics of the area so that I can understand conversations that mention the topic'
(5) 'I need to have wide knowledge of the area and its related issues, but I do not need to employ the skill myself.'
(10) 'I must have in depth knowledge of the topic and its related issues and must apply it to real world problems on a regular basis.'

Question 7: For each of the specific 'Software Engineering' topics below, indicate the level of expertise needed in the area to *perform well in your job requirements*. Use the same interpretations as in Question 6.

Question 8: For each of the specific Software Engineering topics given in the previous question indicate your level of expertise in the area *upon graduation from college*. Use the following interpretations for levels 1, 5, and 10:

(1) 'I was vaguely familiar with the definition of the topic, but I did not study or apply the topic in any of my college classes.'
(5) 'I studied the topic and had wide knowledge of the topic and its related issues, but I never applied the topic to a real problem in any of my classes.'
(10) 'I had in depth knowledge of the topic and its related issues and had applied it to real world problems during my college education.'

The general skill areas and software engineering topics are listed in Table 5.

Study results and analysis

Question 6: General skill categories

Question 6 asks respondents about 'soft skills', which are important in software engineering [17] for any capstone project incorporating team-oriented and customer-facing activities. The data in Fig. 2 show a breakdown of Question 6 responses by graduation date.

The responses in Fig. 2 should be read left-to-right across general skill category, with each grouping showing responses for current students, students graduated within the past year, and students who graduated more than one year ago. The interesting pattern here is that current students are quite confident in their soft skills (purple bars), while students fresh out (magenta bars) are not—but then rebound (green bars) after gaining some experience.

Table 5. General skill areas (Question 6) and software engineering topics (Questions 7 and 8) on the survey

General skill areas		Software engineering topic areas		
Problem solving skills	Risk management	Integrated development environments (IDE)	Test-driven development (TDD)	Project management
Programming ability	Estimation	Defensive programming	Defect tracking	Task planning and sequencing
Current technologies	Teaming	Code reviews	Quality planning	Estimation
SE knowledge	Communication	Software builds	Postmortem reviews	Software metrics
Software processes	Technical writing	Use case analysis	Test types (Pilot, alpha, beta, etc.)	Software cost estimation
Produce quality deliverables	Giving presentations	Software packaging and deployment	Requirements elicitation	Configuration management
Business considerations	Leadership	Human computer interaction/User interfaces	Writing requirements	Release management
Impact of open source	Conflict resolution	Refactoring	Data modeling	Deployment planning
Organization	Negotiation	Software maintenance	Behavioral modeling	Risk management
Time management	Professionalism and ethics	Unit testing	Flow modeling	Capability Maturity Model (CMM)

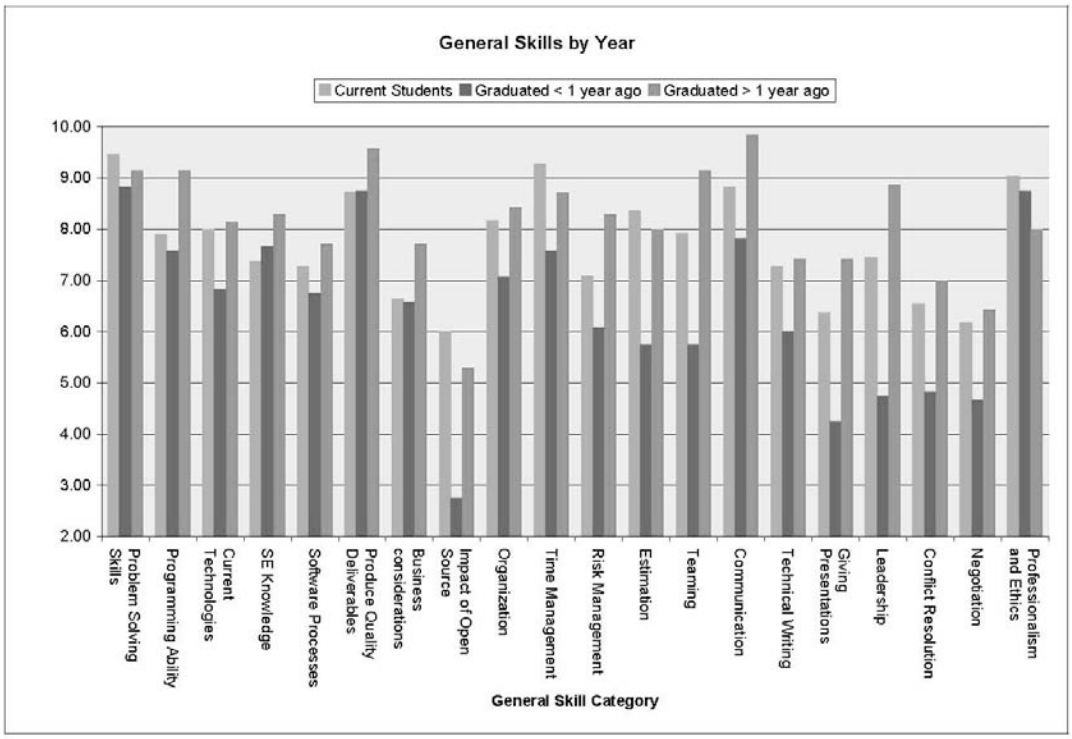

Fig. 2. Average responses for Question 6 broken down by graduation year.

We do not have conclusive evidence as to why this pattern emerges. One hypothesis is that recently graduated students may be overwhelmed by the structure of software development organizations, and feel ill-equipped to handle the interactions. However, this could be the case in many disciplines, and not specific to software engineering. Additional study, such as interviewing or comparison against non-Enterprise student baselines is needed to confirm this hypothesis or gain insight to propose and alternative.

One interesting data point to point out is the decline in 'Professionalism and Ethics' (right side of Fig. 2). This could suggest disillusionment with the practice as one gains experience, and suggests that professional communities still have work to do to mature the profession.

Questions 7 and 8: Software engineering topics

Questions 7 and 8 are of particular interest to this paper as they ask about best practices. We show the responses along two lines of interest. First by question and graduation year, as in Fig.

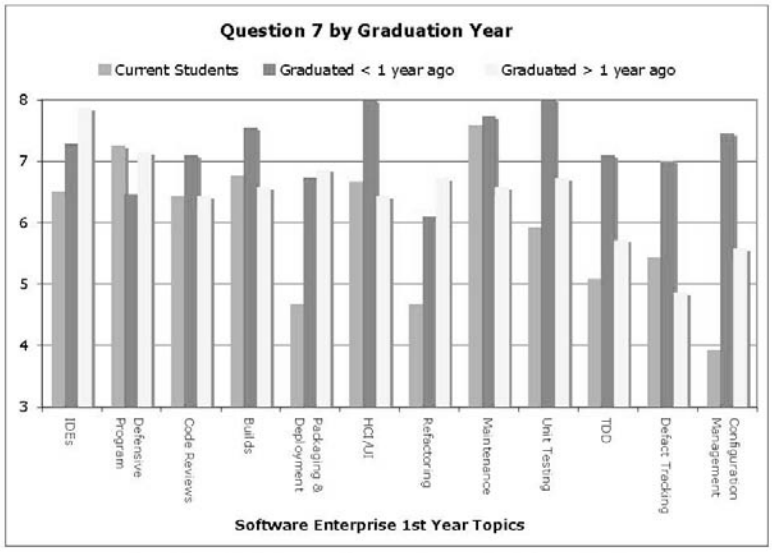

Fig. 3. Average responses for Question 7 for Software Enterprise Year 1 topics.

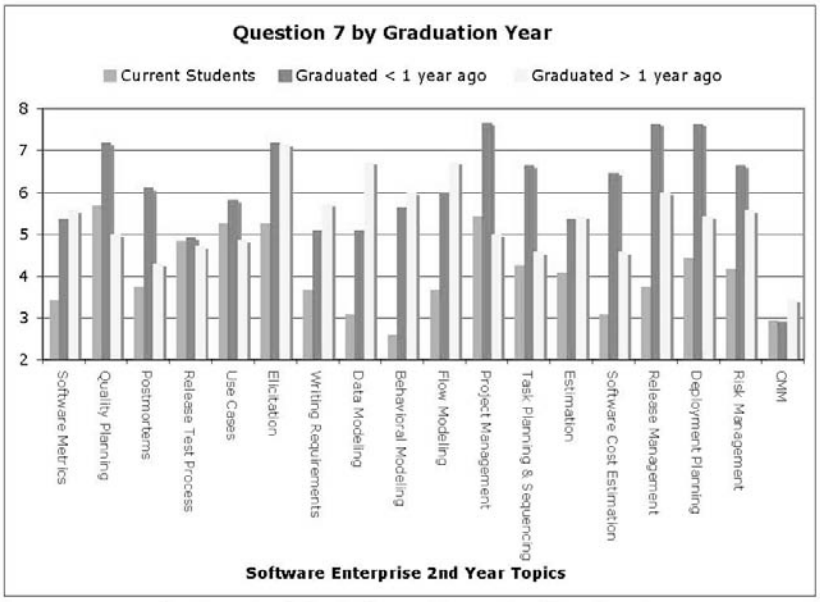

Fig. 4. Average responses for Question 7 for Software Enterprise Year 2 topics.

2 for Question 6. Then we look at the responses to each question transposed against each other.

Question 7 asks participants what practices are needed in their jobs. Figures 3 and 4 show Question 7 responses for topics related to the first and second years of the Software Enterprise, respectively.

In our view, the most noticeable trend in Fig. 3 is along the right side of the chart, in categories (unit test, TDD, defect tracking, CM) that represent low-level developer practices, typically not emphasized in computing curricula. Student perceptions (blue bars) of the need for these practices are low, but once they graduate, the perception drastically changes in the first year (red bars). After the first year of employment, the perception retreats, and is probably closer to actual need (though this study does not attempt to verify this).

Figure 4 shows Question 7 responses for second year Enterprise topics. Many of these categories follow the pattern of the right side of Fig. 3; current students lack an appreciation of the need for the skill, but change perspective after graduation. This can be seen by the deltas between the blue and red bars in almost every category. Also interesting is that unlike other topics, some of the topics continued to increase in perceived need after the first year of employment. These skills, in the center Fig. 4, are geared toward analysis modeling skills instead of the quality, requirements, and

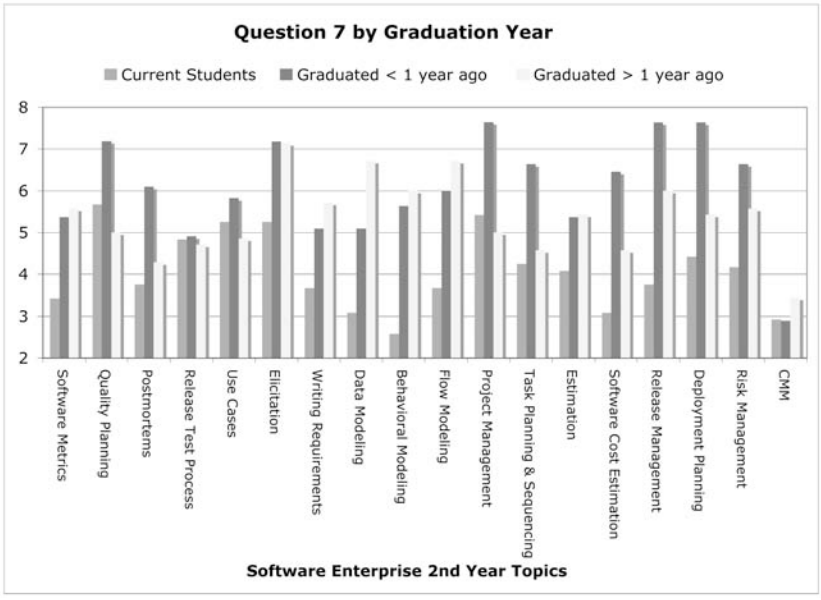

Fig. 5. Average responses for Question 8 for Software Enterprise Year 1 topics.

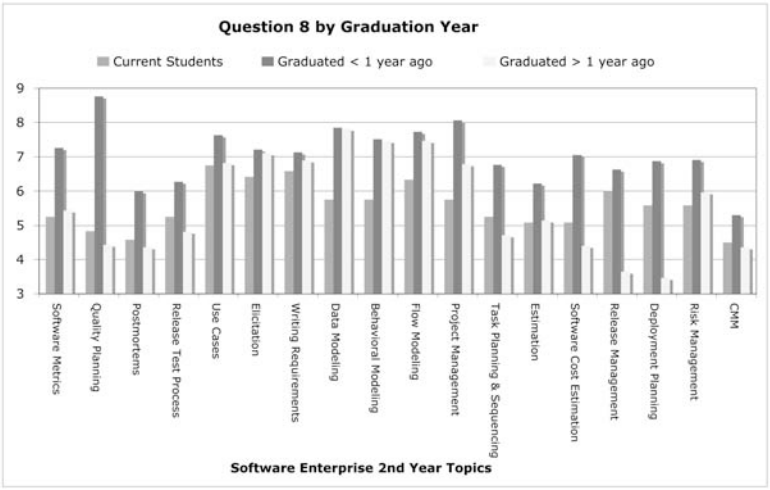

Fig. 6. Average responses for Question 8 for Software Enterprise Year 2 topics.

project management skills represented in the rest of the figure.

The same breakdown is shown for Question 8 in Figs 5 and 6.

Question 8 asks participants about their level of expertise upon graduating from college. Perhaps most noticeable in the data is that graduates who have just completing their degrees feel good about how well prepared they are; but after the first year they change perception quite a bit. The news is not all bad however; greater than 1 year levels (yellow bars) are better than current student perceptions in many second year topics. First year and related development-oriented topics do not fare as well.

Taken together, Figs 2–6 on the survey questions suggest that new graduates feel overwhelmed by the soft skill requirements in their new employment experience, but feel confident in their software engineering skills. Anecdotally, conversations with former students back up these findings. These alums express amazement at 'how the world works', while also expressing a gratitude for being prepared with applied knowledge in the full range of software engineering skills. To verify these findings, a longitudinal study is needed over time coupled with baseline data from non-Enterprise graduates.

The principal issue encountered was the lack of existing baseline data with which to compare our results. It would be useful to know how Division of Computing Studies students performed in their first industry positions after graduation, for graduates prior to the Enterprise sequence introduction or who did not enroll in the Enterprise for some reason. This would tell us how much better we are doing. We were also unable to get industry mentor respondents to participate in the survey, which would have given us a comparative basis for evaluating how Enterprise students do versus traditional Computer Science and Software Engineering graduates.

RELATED WORK

Most academic computing programs now offer a significant software design and construction project course. It is not within the scope of this paper to present a comprehensive overview to software project course offerings. We call out some specific programs with exemplary project offerings that include a significant process component into the teaching and learning lifecycle.

The Software Development Studio component of the Professional Master's program at Carnegie Mellon University (CMU) [22]: The Software Development Studio at CMU is a seminal program in project-oriented software coursework. The Studio puts graduate students in a terminal degree program through a multi-semester project experience covering the full range of software process activities. The Software Enterprise shares the multi-semester approach with an emphasis on soft-skill development with the Studio. The Enterprise, however, introduces the software phases in reverse order, and emphasizes soft-skills development through multi-year structured student collaborations.

The Software Development Laboratory at the Milwaukee School of Engineering (MSOE) [22] also reverses the process phase sequence. The author acknowledges the difficulty that newer students have grasping process and soft-skills concepts, and therefore students are led from 'grave to cradle' through process phases. Unfortunately a further description of the utility of this approach is not provided. The author also discusses the issue of student turnover, or project continuity, and describes a pre-course for seniors that prepares them for the project sequence. This course includes mentoring activities from project enrollees, shared advice on the project, and basic skills preparation.

This is a model we are looking to replicate in our Year 1, first semester Tools offering.

Auburn University emphasizes the capstone experience as a teaching and learning vehicle [24]. The authors provide a strong argument that student teams cannot be thrown together into teams at the end of their undergraduate experience, and then be expected to know how to assimilate all of the software engineering concepts from the previous semesters and apply them on a real-world project. We enthusiastically agree with the authors on this point, and this is why the Enterprise uses the 'Instructor-facilitated, Learner-centered' delivery model discussed in the section 'Pedagogical model.' Furthermore we share some of the detailed logistical concerns expressed by the authors regarding how to form teams and distribute tasks.

Wilde et al. [25] suggests that traditional classroom instruction in software evolution and process requires a significant hands-on component for students to gain true understanding. The authors implemented a model called GUMP at the University of West Florida where student teams evolve an existing software product for the capstone experience. The authors briefly discuss some experiences of exposing students to Agile methods (specifically XP) to '. . .try to provide speed while still preserving reasonable quality standards', but this exposure is not described at the best practice level. The emphasis on existing software products and learning process through evolution is an excellent approach, though the Enterprise combines this with a more distinctive real-world emphasis than the relatively controlled environment used here.

Hazzan and Dubinsky go beyond reporting on capstone experiences [12] to a methodology (SDM) based on fourteen principles [13]. Several of these principles are shared in the Enterprise, such as 'reflection' (principle 4). The general approach is inspired by *active learning*, which is philosophically aligned with the hands-on, 'learn-by-doing' approach of the Enterprise. It may be no coincidence that the authors' approach evolved out of capstone experiences based on extreme programming (XP). There is suitable anecdotal evidence in the community that capstone experiences leveraging Agile methods *must* use an active learning approach, and this should be followed up with sustainable assessment (more sustainable than we have been able to present in this paper). The SDM method is a good start philosophically, but needs to be operationalized into an alternative curricular model repeatable in the academia

The viewpoints on industry-preparedness in the academic community still vary widely. This is reflected in the range of parameterizations and emphases on capstone projects. Some capstone projects are more structured than others; some encourage more industry connectivity than others; and most vary on the way teams are formed. So although there is some common understanding of the motivations and underlying features of the capstone, there is little agreement on the machinery to better prepare students for industry. Only a few papers take on this question directly. Cowling [6] summarizes a benchmark framework for software engineers at both the undergraduate and graduate level in the UK. The author suggests that recent efforts on defining the knowledge base of the profession, such as the SWEBOK [5], are worthwhile but must be complemented by a similar definition of what graduating students are able to practice. A generic framework derived from more mature engineering fields is adapted to software engineering, with the product model levels identified: Conceptual, Determinable, and Physical. However, the author goes on to admit that, in contrast to other engineering fields, software engineers treat construction '. . . as an activity that is conducted by engineers rather than technicians' and concludes that '. . . SE programmes at either level will still need to achieve a reasonable level of competence at actually undertaking construction.' We agree with this statement and it is the reason that we emphasize best practices at a detail level. Denning, in a recent essay [7], bemoans that software engineering and the computer science academic community as a whole has not learned its lessons well. He advocates a more blended approach combining programming, algorithms and design, and defect correction, with an assessment model based on increasing levels of competence. We agree in principle with the blending idea, though we believe the devil is in the details. An enormous amount of work and study is required to pull this off effectively, and it represents a significant shift from how computing is currently taught. He also states that programming is not the main problem, and here we disagree. Quality is an attribute of a system that is injected during construction. 'Soft skills' are fine, we teach a significant number of them in the Enterprise (see Table 3), and even those applying lightweight process models apply key best practices for project management. But, the emphasis should always be on the code. We believe the best way to ingrain both soft skills and programming ('hard') skills is not just to see how they work, but how to apply them. An earned-value (or 'burn') chart that shows the students how *their* project is falling behind together with a risk analysis model that tells them how and why is far more effective than simply giving them a homework exercise that has them calculate earned-value on a problem *they do not own*.

SUMMARY AND FUTURE DIRECTIONS

The Software Enterprise has rapidly matured into a differentiator for the Division of Computing Studies at Arizona State University's Polytechnic

campus. It is a required component of the Applied Computer Science program and embodies the hands-on learning style of the Polytechnic campus. As our program heads toward an initial accreditation review, we hope to collect better baseline data about our students and other programs around the country to see if this is an effective and sustainable model. Problem-centered learning is a natural teaching and learning style for the Polytechnic as it ensures that students not only acquire the skills required to be successful in local and national industry; more importantly, it teaches them how to acquire or update such skills in a technologically changing and challenging world. Our most significant outcome is that we have reached an initial plateau where our graduating students are better prepared for their first positions as Software Engineers due to the hands-on, iterative pedagogical model we have designed and implemented for the Enterprise.

The Software Enterprise reflects our belief that engineering know-how is acquired, applied, and durable through its contextual and repeated practice, and not through a traditional teaching and learning model. Instead of setting a broad foundation and then driving deep into narrow area of specialty, the Enterprise instead looks at the emergent expertise gained by junior professionals to create a new pedagogical model for software engineering that mimics exposure and teamwork found in industry. We advocate that this is the best model for attaining program objectives in software engineering.

We agree with Cowling [6] that significant effort has been spent identifying a mature body of knowledge for the profession and work is not yet complete in moving this body of knowledge into the classroom. But while this work is underway, we must look forward to the next steps. This knowledge is useless without a proper mechanism by which to raise learner competencies to an applied level. Our view is in line with the view espoused by Parnas [21] almost a decade ago. Parnas indicated that a body of knowledge was just one of several things needed for future software engineers to learn. Two others revolved around applied knowledge. Junior professionals learn principally through mentorship; as software engineering educators we must scale the mentorship model to the masses through more engaging, practice-oriented teaching techniques. This paper has suggested we can start with the capstone experience.

REFERENCES

1. K., Beck, *Extreme Programming Explained—Embrace Change,* Addison-Wesley, Boston, (2000).
2. B. Boehm, A spiral model of software development and enhancement, *IEEE Computer*, May 1998, pp. 61–72.
3. B. Boehm, A. Egyed, D. Port, A. Shah, J. Kwan, and R. Madachy, A stakeholder win–win approach to software engineering education, *Annals of Software Engineering*, **6**, 1998, pp. 295–321.
4. J. Borstler, D. Carrington, G. Hislop, S. Lisack, K. Olsen, and L. Williams, (2002). Teaching PSP: Challenges & lessons learned, *IEEE Computer,* September/October 2002, pp. 42–48.
5. P. Bourque and R. Dupuis (eds.), *Guide to the Software Engineering Body of Knowledge*, IEEE CS Press, Los Alamitos, CA, (2001).
6. A. Cowling, What should graduating software engineers be able to do? *Proceedings of the 16th Conference on Software Engineering Education and Training* (CSEET '03), Madrid, Spain, (2003).
7. P. J. Denning, The field of programmers myth, *Communications of the ACM*, **47**(7), 2004, pp. 15–20.
8. K. Gary, H. Koehnemann, and B. Gannod, The software enterprise: Facilitating the industry preparedness of software engineers, *Proceedings of the National Conference of the American Society of Engineering Education* (ASEE '06), Chicago, IL, (2006).
9. K. Gary, G. Gannod, H. Koehnemann, and M. B. Blake, Education future software professionals on outsourced software development, *Proceedings of the National Conference of the American Society of Engineering Education* (ASEE '05), Portland, OR, (2005).
10. K. Gary, G. Gannod, H. Koehnemann, T. Lindquist, and R. Whitehouse, WIP: The software enterprise, *Frontiers in Education* (FIE '05), Indianapolis, IN, 2005.
11. K. Gary, The software enterprise: Preparing industry-ready software engineers, *Software Engineering: Effective Teaching and Learning Approaches and Practices*, Ellis, H. (ed.), IDEA Publishing Group, (2007).
12. O. Hazzan, and Y. Dubinsky, Teaching a software development methodology: The case of extreme programming, *Proceedings of the 16th Conference on Software Engineering Education and Training* (CSEET '03), Madrid, Spain, (2003).
13. O. Hazzan, and Y. Dubinsky, Teaching framework for software development methods, *Proceedings of the International Conference on Software Engineering* (ICSE '06), Shanghai, China, (2006).
14. T. Hilburn, and W. Humphrey, Teaching teamwork, *IEEE Computer,* September/October 2002 pp. 72–77.
15. W. S. Humphrey, *PSP: A Self-Improvement Process for Software Engineers*, Addison-Wesley, Boston, (2005).
16. W. S. Humphrey *Introduction to the Team Software Process*, Addison-Wesley, Boston, (2000).
17. IEEE/ACM Joint Task Force on Computing Curricula, *Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*, (August 2004).
18. J. Kawakita, *The Original KJ Method*, Kawakita Research Institute, Tokyo, (1991).
19. P. Kruchten, *The Rational Unified Process—An Introduction*, 2nd edn, Addison-Wesley, Boston, (2000).

20. T. Lethbridge, What knowledge is important to a software professional, *IEEE Computer*, May, 2000.
21. D. L. Parnas, Software engineering programs are not computer science programs, *IEEE Software*, November/December, 1999.
22. M. Sebern, The software development laboratory: Incorporating industrial practice in an academic environment, *Proceedings of the 15th Conference on Software Engineering Education and Training* (CSEET '02), Covington, KY, February, (2002).
23. J. E. Tomayko, Carnegie Mellon′s software development studio: A five year retrospective, *Proceedings of the 9th Conference on Software Engineering Education* (CSEET '96), Daytona Beach, FL, (1996).
24. D. Umphress, T. Hendrix, and J. Cross, Software process in the classroom: The capstone experience, *IEEE Computer,* September/October, 2002, pp. 78–81.
25. N. Wilde, L. J. White, L. B. Kerr, D. D. Ewing, and A. Krueger, Some experiences with evolution and process-focused projects, *Proceedings of the 16th Conference on Software Engineering Education and Training* (CSEET '03), Madrid, Spain, (2003).

Kevin Gary is an Assistant Professor in the Division of Computing Studies at Arizona State University. His areas of scholarship include software engineering, particularly Agile methods and open source software, and enterprise application development. Kevin is a co-Director of LEAD (Laboratory for Enterprise Application Development) and DEAC (Distributed and Enterprise Applications Consortium) at ASU. He is an active participant in the open source Image-guided Surgical Toolkit (IGSTK) project and is actively engaged in applied research and industry consulting. His exposure to industry best practices, particularly in Agile software development enterprises with junior developers, led him to devise the Software Enterprise.