Team-Oriented Software Practicum

Jeffrey C. Schlimmer, Justin B. Fletcher, Leonard A. Hermens

Abstract—
New computer science graduates are inadequately prepared to apply their abstract knowledge, and they have rarely worked in the teams required in modern business. To remedy these shortcomings and alleviate the low motivation that often accompanies the first years of college, we proposed that a pilot group of entering freshmen should be formed into a four-year experimental team. Unlike traditional course-at-a-time approaches, this supplements the existing curriculum by integrating material across courses through team construction of software projects. Under the guidance of faculty and graduate student mentors, students work cooperatively on projects related not just to programming but to the entire lifecycle of software production, from market analysis to revision based on technical support. Initial projects are team-oriented and scaled to the capabilities of entering students while final projects span the product development cycle and involve several semesters of effort. Industrial representatives provide a practical perspective by presenting seminars on special topics and evaluating student projects in light of professional standards. This type of experiment provides the industrial community with students better prepared to face the challenges of professional software development. It also offers consolidated learning, enhanced student retention, significant student-faculty involvement, and the potential to identify learning experiences that may be usefully integrated into existing courses.

Keywords— **Curriculum development, freshman programs, multi-semester curriculum, software engineering education, software practicum, student teams, teamwork.**

I. Introduction

Graduates from today's universities are ill-prepared to contribute to the modern work force. Specifically, professionals from the software industry have revealed several problems with new graduates which may be roughly grouped into two categories: practical perspectives and social working skills.

In the former category, observers note that new graduates lack practical competence, cannot build useful systems, and that industry must often put graduates through year-long internal company training programs [5]. For new computer science graduates, these deficiencies mean they do not understand the trade-offs that arise in practical situations. They produce code that is elegant at the expense of efficiency and is difficult to maintain. They typically place little importance on understanding the users' task area, leading to software that doesn't meet users' needs and is difficult for the user to use effectively. New graduates are also frequently unaware of current software practices (like code walk-throughs) and have difficulty fitting into an organized framework of software production. In short, they need a more even balance between practical and theoretical knowledge [7].

In the latter category, business representatives note that students are often unaccustomed to working in teams, they are poor at expressing ideas verbally and in writing, and they often have unteachable attitudes. For instance, Phil Condit, president of the Boeing Company, stated in his 1993 Lanning Distinguished Lecture [4] that the top two desirable capabilities in a modern worker are communication skills and the ability to work well in groups. In the software industry, it is estimated that about 50% of a software engineer's time is devoted to working with others in group situations [12], so it is not surprising that communication skills and the ability to cooperate with others are very important attributes.

From the university's point of view, entering students are often "turned off" by the present curriculum since initial course work emphasizes learning in large classes. In general, this represents a significant discouragement for new freshmen, particularly minority and underrepresented students who face significant obstacles in adjusting to the new social and academic demands of university life.

In light of these shortcomings, undergraduate educators should have three particular goals: (a) ensure that students have both a conceptual and a practical understanding of course material, (b) develop students' interpersonal working skills, and (c) raise students' motivation to learn the material and instill good habits early in the curriculum. In computer science education this means that: (a) students learn both the theoretical and the applied sides of programming, (b) that they learn to work together with others to create software effectively, and (c) that they are motivated not just to get their programs working but to learn how to design high-quality software.

The first goal involves the acquisition of both general and technical knowledge. This includes ensuring that students can analyze new problems, compare alternative approaches, and synthesize new solutions. As part of dealing with realistic situations, they should also be able to cope with ambiguity. One technically-oriented objective that needs specific emphasis is that students should be able to apply abstract course material to practical situations. Optimally, the students' skills would be transferable, allowing them to apply their expertise in new situations.

The second goal entails people-oriented skills that are undeniably part of contributing effectively to society. In addition to general skills, like being able to effectively com-

The authors are with the School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164-2752

municate verbally and in writing, students should also be able to learn in a self-directed manner from open-ended, project-oriented situations as well as formal, structured instruction. Beyond these, team skills are also important. Students should be able to cooperate, negotiate, and persuade each other to accomplish group objectives. Ideally, students develop the ability to supervise, instruct, and even mentor each other in areas where one has a strength corresponding to another's weakness.

The final goal involves issues inherently related to developing student interest in the topics of study. Principally, undergraduate students should be sufficiently motivated to apply themselves diligently to their course work and move beyond the tenuous level of academic commitment common in the high school setting. Other logically related goals are recruiting quality students and the retention of capable students, especially minority and underrepresented students.

Consider each of these goals in the context of a computer science education; specifically, students should become skilled in developing meaningful software requirements, comparing alternative algorithmic solutions, and constructing new algorithms when required. They should also be able to handle uncertain requirements with flexibility as well as being able to revise existing code to meet new requirements. To integrate abstract ideas with applied knowledge, they should be adept at balancing abstract elegance for run-time efficiency and maintainability in software. For the sake of transferrable skills, students need to acquire a clear understanding of general programming issues that apply no matter which programming languages they encounter during their careers. Similarly, instead of focusing on current industry practice in software engineering for its own sake, students should be able to use current practice as an example of specific approaches to general issues, allowing them to transfer their skills to new software development paradigms when introduced.

In an undergraduate computer science program, students should be willing to sacrifice not only the late hours often required to successfully complete programming assignments (as will be required by industry to maintain a schedule), but should also be willing to invest in careful study of the principles that lead to clear, reliable, and easy-to-construct software (and consequently save themselves many hours of debugging). The emphasis of this task is not on getting programs to run, but on student understanding of the fundamentals of quality software construction.

The application of these goals to a computer science education implies that students should be able to clearly discuss software requirements and approaches as well as generate precise and readable design and program documentation. They should be able to glean reusable ideas from both software construction projects and from traditional, lecture-oriented classroom instruction. As team members, they should be able to work together both to construct high-quality software and to complement each others' strengths.

II. Potential Impact and Significance

The pilot project described here marries realistic project-oriented instruction with a team organization to address the first two goals, and it provides both short- and long-term motivation to address the third. To provide an intensive experimental setting, the basic approach was to form a small, four-year team of entering freshmen and use assignments in the team environment to supplement their traditional course work. One faculty member and two graduate students mentored the team, leading them through a series of exercises drawn from the lifecycle of software development. To provide realistic assignments and integrate abstract knowledge with practical situations, projects were formulated, guided, and evaluated with the help of software industry professionals.

This team-oriented practicum provides a flexible experimental setting from which prototype activities might be moved into standard curriculum. Because the team is relatively small, ineffective activities can be easily replaced by alternatives. Because the bulk of the team is to be maintained over the duration of an undergraduate education, it is possible to evaluate the merit of freshmen activities from the senior perspective.

The addition of activities embodied in the team-oriented practicum to a course-oriented curriculum offers substantial improvements in at least four areas, and these parallel the goals identified in Section I. First, students are given repeated opportunity to apply, test, and revise their understanding of abstract course material. Second, students are given an opportunity to practice working with other students (in sharp contrast to the classroom situation where the need to evaluate individuals forces students to work by themselves or face charges of cheating). Third, students are given exciting tools and exceptional industrial feedback, leading to higher motivation and retention and, eventually, improved recruiting of new students. Fourth, students are given a stable, multi-semester environment where their efforts in individual courses contribute to their overall understanding. Students often forget material soon after the final exam. However, long-term team projects require application of material from several courses, thus improving the retention of material while providing a framework for learning in subsequent courses. An experimental project was needed to identify effective instructional methods because some of these objectives are at odds with the current academic system.

III. Background and Related Efforts

Traditional approaches within the university for addressing the shortcomings outlined in Section I include incorporating technical writing and practical project class work into the curriculum. To date, these methods have not proven sufficient because they are bound to the demands of individual evaluation (limiting team efforts) and are limited to semester-sized presentations of material (limiting the depth of project work and hampering a systematic understanding of issues). The repeated dissatisfaction expressed by practicing software professionals about new graduates sug-

gests that adding more of the same type of activities to the traditional subject-at-a-time approach is not likely to be sufficient.

The tradition of requiring senior-level, capstone design course sequences is a focused effort to provide significant opportunity for bringing together material from several courses and may allow for significant depth in a project. However, this appears to be too little, too late: projects are often only a single semester in length, students do not benefit from the integration of ideas and practice until the end of their studies, and team-orientation is often undermined by scholastic competition for grades.

Wulf [19] offers a significantly different approach that is designed to eliminate the tendency of undergraduate computer science students to become sidetracked in hours of needless debugging and to break the habit of assigning short programs that students must construct from scratch. Wulf suggests following the model of laboratory training used in other scientific disciplines. Instead of giving students assignments to complete on their own (possibly poor) time schedules, they are given specific, tightly-constrained problems to solve during a specific time, under the supervision of the instructor, and in direct correlation to the material currently being presented in the main lecture section of the course. Termed "closed laboratories," the set of exercises directs the student toward appropriate solutions and limits the amount of time students waste trying to implement inherently poor algorithms. This method does give the students more time to consider the underlying issues, but the structure of the laboratory assignments may discourage projects of any significant detail or size and does not necessarily guarantee integration of material across courses. One may be able to incorporate some elements of teamwork, but because the laboratory assignment needs to be relatively well-structured and similar for each group of students, it apparently precludes efforts by teams of more than two or three.

Marbury, Barnes, Lawsine, and Nicholson [11] offer a more radical approach that seriously restructures the current semester-sized division of material. While not completely discarding the traditional university course structure, most of the material in an upper-class student's major would be taught by a single faculty mentor over the course of a series of semesters. Modeled after a one-room school house, it builds on the idea of a single instructor who provides continuity to help overcome the distracting effects of an "expanding curriculum" and can help students integrate material covered over several semesters. Of course, such an approach requires unprecedented curriculum revision. It also requires considerable investment on the part of each professor who serves as the school room instructor. There is an increased risk associated with poor instructors because one-room students will not be exposed to the balancing effects of courses taken from a variety of professors. This approach could support realistic projects with significant depth, and potentially it could train students to work collectively as a team.

Both of these innovative approaches focus on the goal of allowing students to test, apply, and evaluate course material in an integrated manner. However, they do not explicitly provide for team-oriented learning, and neither directly addresses the goal of improving student motivation, retention, and recruiting. In contrast, the team-oriented software practicum described here offers mechanisms for integration, consistency, teamwork, and motivation.

A new multi-year program at Old Dominion University called the Computer Productivity Initiative (CPI) project [10] involves undergraduates in software engineering at the sophomore, junior, and senior level. Sophomores in the program will take a programming/problem solving course, juniors will typically take a computer in society course, and seniors will involve themselves in a traditional two-semester software engineering/design sequence. The senior-level course requires student groups to approach a broad problem description, extract and formulate the software requirements, then plan the project as one would if they were working in the software industry.

In contrast to many introductory software engineering courses, the CPI project features the use of a computer-aided software engineering (CASE) tool. In particular, the courses use a "decision-based hyper-multi-media" CASE tool that was built to aid the curriculum planning and development process. Furthermore, the CPI project emphasizes real-world issues, and encourages initiative to find solutions to relevant software engineering problems. For example, project planning, project management, cost estimation, and market research are all part of the senior courses. (Not all university software engineering courses offer this coverage in the introductory courses.)

Motivated by similar concerns, Denning and colleagues [6] have founded a center for improving engineering education. Termed the "Center for the New Engineer" (CNE), their effort encompasses ten separate projects. Like the CPI project, several CNE projects focus on incorporating new technology (like collaborative software) into student training. Other CNE projects are studying more fundamental changes to the university's structure, including forming tightly coupled consortia of companies and academic departments jointly responsible for instruction, and changing the basic work of a faculty member from lecturing and grading to serving as a supervisor for student invention. All these projects share some common motivations. Principal is the concern that faculty have abandoned teaching for research as a consequence of shifts in the public's understanding of the mission of universities. Besides calling for greater commitment, the general strategy behind the projects is to teach knowledge that leads to effective action, encourage students to assume responsibility for their education, and identify approaches that integrate research and teaching.

The practicum we describe in this paper encompasses many of the concepts found in the CPI and CNE projects, but we are not placing any special teaching role on advanced computing technology. The practicum does seek to combine practical knowledge of state-of-the-art tools and practices with opportunities to build communication and

team skills. Industry evaluation of student projects over four years serves to calibrate student expectation and develop professional-quality skills.

IV. The First Year

Freshmen selected for participation in a team are likely ill-prepared to begin work immediately on a sophisticated, long-term software development project because they do not have sufficient background in basic algorithms, data structures, or programming languages. In fact, this is precisely what the traditional, course-oriented undergraduate system is designed to teach. However, incoming freshmen are well prepared to develop team skills because much of high school is effectively a social experience, and team projects foster a high degree of friendship [8,16]. For these reasons, the focus for the first two semesters was on team-oriented problem solving and non-coding topics within the software development cycle.

A practicum team was formed in the fall of 1992 at Washington State University. Students enrolled in the first semester programming course for Computer Science majors were invited to participate. These students spent their first few weeks in three activities aimed at fostering teamwork and helping them understand their unique abilities. First, we held an informal barbecue for any student interested in the practicum. After eating, we played three short, non-competitive games that require cooperation between two or more people (cf. Weinstein & Goodman [18]). A few days later, students were assigned to four person teams to play the game SimCity, a computer simulation of city planning and economics. To create a situation where cooperation was essential, each student was assigned a particular duty: one student made all decisions about power and transportation, one made all zoning decisions, one dictated the locations of police and fire stations, and one sat at the keyboard as mayor, entering all commands and setting the tax rate. As we watched the teams, we were able to observe team member's personality traits which would later guide us when working with the students.

In parallel with these activities, each student took the Myers-Briggs Type Indicator [2]. Team personalities are well balanced between type I (introvert) and type E (extrovert) with five of the former and four of the latter. Other dimensions are slanted towards type N (intuitive) instead of type S (sensing) [six to three], towards type P (perceptive) instead of type J (judging) [seven to two] and towards type T (thinking) instead of type F (feeling) [eight to one]. A study reporting a sample of American engineering freshmen [14] shows a balance between personality types in all but the thinking versus feeling dimension. In that study as with our freshmen, the majority of the students are type T.

We supplemented the Myers-Briggs Type Indicator by working on and discussing exercises from an occupational interest text during our first three team meetings [1].

This initial phase serves two purposes. First, it exposes each student to the team-orientation of the practicum. Second, it serves as an important self-selection for students and helps identify those that are willing to be committed to the practicum. In addition, to evaluate their ability and motivation, each student was given a set of logic puzzles to complete and was asked to write a paragraph describing why they wanted to be on the team. Twelve students attended the barbecue, but by the time we finished with the initial three weeks, only nine students remained. This was a bit larger than the target team size of seven students, but it allowed for attrition of one or two students without drastically reducing the team size. One consequence of this self-selection process is that it ensures that the members of the team are highly committed individuals. However, they are not all exceptionally gifted academically; their average grade in the first two programming courses was B.

The second phase of the first semester focused on an aspect of software development that does not require coding expertise: market analysis. The paradigm we followed for this topic will be repeated throughout the practicum, namely: student preparation using a textbook, discussion, seminars by industry representatives working in the area, work on a related project, and evaluation by industry representatives. Specifically, in the fall of 1992, students on the team read and discussed a marketing text [9] during the team's weekly meeting. Representatives from two regional software companies each visited WSU and presented seminars on software market analysis. One company enthusiastically involved the team in a current product analysis by disclosing marketing information to the team and providing copies of competitor software for the students to critically evaluate. The team prepared a marketing requirements summary which was subsequently evaluated by four employees of that company. The employees evaluated the team's summary as quite thorough but awkwardly written and somewhat naive. To refine the team's writing ability, the company formulated a supplementary project to evaluate one competitor's product in more detail. During this time, the team also hosted a seminar on team dynamics and took a field trip to another regional software company.

The team's second semester focused on software quality assurance. Again, team members prepared for industry seminars by studying a text on testing [13]. As an exercise, the students did a code walk-through and a code review on one of their solutions to a class assignment. We also read and discussed *The Mythical Man Month* [3], a dated but enlightening text on the realities of software engineering. The team hosted four seminars on software quality assurance. As projects, the team beta tested MS-DOS 6.0 and a prerelease version of a software networking product. In both cases, test reports generated by the team were judged by the developers as typical of user defect reports, somewhat short of the quality expected of software test engineers, and nowhere near the volume of data expected. The students accurately identified defects but not nearly enough them.

Neither software marketing or software quality are traditionally covered in early undergraduate computer science courses and may never be covered, depending on the departmental curriculum and the student's course selections. The first topic will benefit the students' outlook, helping

them eradicate the notion that problems will always be well defined. The second topic will certainly serve them well by training them to test their software projects for other courses.

V. Future Years

The third and fourth semesters will follow the same basic prepare-seminar-project-evaluate cycle because it has been successful and popular with students and our industry contacts. In Fall 1993, we are scheduled to study project management and, in Spring 1994, issues in software maintenance. Appendix B lists the texts used and planned for the duration of the practicum.

The final four semesters are currently reserved for producing a piece of production quality, marketable software incorporating all aspects of a waterfall software lifecycle: analysis, design, implementation, testing, and maintenance. Some representatives from regional software companies have expressed a keen interest in contributing to the formulation and evaluation of projects at this stage in development.

Over the long term, the team will run more smoothly if membership retention is high, but dropout is inevitable. The team had nine students its first semester but only seven its second. One student changed majors, and one withdrew from the university. Following the second semester, each of the seven remaining students stated that they intend to remain on the team, but new members will eventually have to be added if the team is to remain at a reasonable size. If members do not return, our plan is to have an open enrollment in the fall, inviting interested students to participate in social, team-oriented activities like those used at the beginning of the first semester. Those that pass this self-selection test will be welcomed onto the team.

VI. Meshing With Existing Curricula

One benefit of this type of educational experiment is the ease with which it can be coupled to an existing program. Ostensibly, participating in the pilot program does not replace any of the traditional courses in an undergraduate computer science program (though perhaps the extra effort could be counted as fulfilling a technical elective). Rather, the practicum supplements existing studies, providing an opportunity for students to apply, test, and evaluate what they have learned. Administratively, each team member is enrolled in a two-unit independent study course advised by the faculty and graduate student mentors. (Whether these units should be graded or simple pass-fail is unclear; there are compelling arguments on both sides of the issue [17]. For our practicum, the credits were graded pass-fail.)

However, the administrative ease of implementation belies the significant faculty involvement required. Consequently, it is best viewed as a vehicle for rapid experimentation. It allows for testing learning activities that expose the students to those aspects of the software development process that are currently neglected early in most curricula.

VII. Mentoring

The overall responsibility for the successful operation of the team lies with the faculty and graduate student mentors. In addition to arranging the logistics of visiting industry representatives and guiding the exploration of projects, the mentors must help troubleshoot individuals' course/study problems. This is essential if the team members are to excel at team projects as well as in their regular course work. (The latter is the basis for their fundamental understanding of software construction.) To help forestall negative group dynamics, the faculty and graduate student mentors must be available for confidential discussions to help treat interpersonal difficulties that will (and did) arise. Rather than establishing typical restrictive office hours, mentors were available as needed; team members were encouraged to discuss topics of interest freely.

During the first semester of the practicum, mentoring proved difficult to implement effectively. The students were initially reluctant to discuss their standing in classes. After realizing that one of the students had considerable difficulty with the first programming course, the graduate student mentor made a concerted effort to meet with the student in a non-critical, open environment. The students also gradually began convening several minutes before the official meeting time to discuss class assignments with the graduate student mentor. During the second semester the team dynamics began to open up as they became more familiar with a team environment. As a result, there were no significant interpersonal problems during the first year.

As the first instance of the practicum proves even more productive, it may be appropriate to begin a second team, overlapping the first. This would allow for considerable peer advising from upper to lower classmen as well as sharing the relatively expensive industry seminar resources.

VIII. Pilot Practicum Implementation Resources

There are two major types of resources needed for a successful implementation of the team-oriented software practicum: equipment and expertise. As team members, students need a common computing environment. Both the marketing analysis and software testing projects required computing hardware, and future semesters will require more computer use. In many cases, centralized, shared computing is too inflexible for team projects. For instance, the marketing project required experiments with modem systems, and the beta testing required installing potentially unreliable system software. Thus far, students have used their own personal computers, but only three of the members had computers during the first year. For maximum flexibility, a specialized laboratory should be established to facilitate computer access, especially where specific hardware is required.

Expertise comes from industry representatives and the faculty and graduate student mentors. Both the faculty and graduate student mentors for the initial practicum have significant industry experience. While turnover is expected with graduate student mentors, an effort will be made to select experienced mentors, allowing them to pass

on their skills in a fashion that would otherwise not be possible in the traditional academic environment. Thus far, the faculty's involvement has been supported by university start-up funds, and industry representatives have provided their own travel support. (In one case, a speaker could not visit because of insufficient funds.) One quarter-time teaching assistantship was allotted for the graduate student mentor and another graduate student volunteered for the first semester. Travel funds, funding for laboratory machines, and a stipend for a full-time teaching assistantship are useful to facilitate the program.

IX. Assessment and Evaluation

A. Evaluation Metrics

The effectiveness of the team-oriented software practicum will be difficult to evaluate precisely. Nevertheless, there are four short-term metrics that may be used to determine the value of the practicum. First, industry's evaluation of the team's projects will be a strong indicator of the overall quality of the practicum's affects; professional quality projects from beginning undergraduates would be a strong vote of confidence. Second, evaluations from internships will also indicate the value of the practicum and represents another way to measure whether industry's justified dissatisfaction with new graduates is being addressed. Third, evaluations from the students themselves may attest to the worth of the practicum. These latter evaluations are similar to those used for internships and include: student expectations, identification of specific skill development opportunities, comparison with course-oriented learning, specific long term educational/occupational goals, and overall satisfaction with the degree earned. Fourth, periodic review by peer educators can help establish if the practicum is maintaining the balance of abstract learning and practical learning without overemphasizing either. This metric is already being evaluated by several computer science educators at other institutions. The third metric will be evaluated by initially collecting information from the students using a form similar to the one in Appendix A and by reevaluating the information at the end of each semester.

B. Evaluation of The First Year

The team performed three projects in its first year for two different software companies. Both companies have responded enthusiastically to the practicum's mission, and both are willing to participate in subsequent projects. Although we do not have a quantitative measure of success in this category, the results are quite favorable.

Internships as an evaluation metric was somewhat of a surprise. Only two of the seven students were even interested in an internship following their freshman year. Most wanted to return to their home town, minimize their living expenses, and work at a well-paying job unrelated to their degree. Three companies expressed an interest in hiring one of the team members, and one actually did so. According to the interviewer, the team member was hired because he had software testing experience; something no other candidate had. Software companies do not seem to invest considerable energy developing internship positions for freshmen students. We expect that team members' interest in a career-related summer position to increase as they approach graduation.

Student evaluations of the practicum were uniformly positive. We used a university-wide teaching evaluation form of 25 forced-choice questions on a 5-point scale. Questions covered categories of student rapport, classroom performance, knowledge of subject, and tests/homework. The overall average of team member's responses for the first semester was 4.74 on a 5-point scale, compared to a department average of 4.03; they rated the second semester at 4.95 compared to a department average of 4.09. For the 26 team meetings the first semester, there were only 9 absences, or 96% attendance; for 31 team meetings the second semester, there were only 3 absences, or 99% attendance.

We also measured student evaluation of the practicum using a personal growth form designed for internships listed in Appendix A. The results of this evaluation are shown in Table 1. The form covers areas of task skills, work habits, work attitudes, and relational skills. At the start of the practicum students rated their achievement, and after the practicum they rated their improvement. Students indicate a nearly uniform (and optimistic) evaluation of improvement as reflected in Table 1, which lists the net growth score for each student on the team. (One row per student. Each entry is the sum of a student's improvement scores. Positive numbers represent improvement. The last row summarizes the evaluation of one student who withdrew from the team after the fall semester.) Noteworthy in this table is the result that students feel that they grew in working attitudes almost twice as much as any other area as a result of participating on the team.

Approximately one year after the inception of the program, students who participated in the software practicum were surveyed about the reasons for their individual motivation and work attitude. The survey asked: "During the previous year on the team, you reported a significant growth in your work attitude. To which of the following do you attribute your personal change?"

- Attending a university
- Participation in sporting activities
- Participation in other outside activities
- Membership in the Software Team Practicum
- Expectations of performance within the academic environment
- Parental pressure
- Peer pressure

The students were asked to rank each item from 7 (most important) to 1 (least important), and optionally list other reasons and rank them.

The results shown in Table 2 indicate that the software practicum is a significant reason for the positive work attitude. Outside activities, attending a university, and the students' general expectation of performance in an academic environment were also significant factors. One student noted that "personal pressure" and the opportunity

Fall 92				Fall 92 through Spring 93			
Task Perf	Work Habits	Attitudes	Relations	Task Perf	Work Habits	Attitudes	Relations
2	12	22	9	10	12	22	8
6	3	10	5	8	4	14	n/a
1	1	1	2	0	1	4	3
2	3	6	2	3	2	3	8
3	4	11	6	-1	2	7	2
2	1	5	3	5	3	7	6
4	1	3	3	1	0	3	2
1	10	18	13				
2.6	4.4	9.5	5.4	3.7	3.4	8.6	4.8

Table 1: Student improvement

Question: To which of the following do you attribute your personal growth in your work attitude?	
Total	Item
24	Membership in the Software Team Practicum
19	Participation in other outside activities
18	Expectations of performance within the academic environment
18	Attending a university
8	Parental pressure
8	Peer pressure
7	Personal pressure
6	Participation in sporting activities
5	Having a job over the summer

Table 2: Student self-evaluation of personal growth causes

for employment over the summer were among the most significant reasons for his positive growth. Another student added, "I have been very pleased with the practicum, I feel I've learned a lot about project management this semester," and "I'm looking forward to starting the two-year project next year."

X. Conclusions

For students to emerge from four-year university programs with a greater understanding of the software construction process, something fundamentally different must be done in their educational process. An attractive alternative is to form students into long-term software development teams and give them real-world problems that require applying their course work and understanding the trade-offs and practices common in modern software development. The practicum approach does not require revision to existing curriculum, but it allows for experimentation with novel learning activities in a team setting. The first year of an implemented practicum has demonstrated that the concept can expose students to a broader spectrum of professional issues, and it has garnered the enthusiasm of industrial representatives and students alike.

XI. Acknowledgments

Several representatives from regional software companies graciously presented seminars on the software process including Ira Dearing of IBM, Lori Kendall of InterConnections, Chris Knotz of Traveling Software, Inc., Hal Setzer of Battelle Pacific Northwest Laboratory, Eric Straub of Microsoft, and Art Walton of IBM. Jack Hagemeister of Washington State University presented a seminar on computer-aided software engineering tools. Ms. Kendall and Marvin Erickson of Battelle Pacific Northwest Laboratory both contributed significantly to the design of the practicum. Mr. Knotz also formulated and coordinated evaluation of projects both semesters. Jon Ochs hosted the team on a field trip to Eureka Software. Robert Warnick of Washington State University allowed the team to use his computing laboratory. Lonnie Dunlap of Washington State University administered and interpreted the results of the Myers-Briggs inventory. Four anonymous reviewers of an earlier draft of this paper identified additional related efforts and suggested clarifications. SimCity is a trademark of Maxis. MS-DOS 6.0 is a trademark of Microsoft.

References

[1] R. Bolles. *What color is your parachute?* Ten Speed, Berkeley, CA, 1992.

[2] K. C. Briggs and I. Briggs Myers. *Myers-Briggs type indicator.* Consulting Psychologists Press, Palo Alto, CA, 1976.

[3] F. Brooks, Jr. *The mythical man month: Essays on software engineering.* Addison-Wesley, Reading, MA, first edition, 1975.

[4] P. Condit. Disciplines without walls: The future of American industry. Lanning Distinguished Lecture, November 1993. Presented at Washington State University, Pullman, WA.

[5] P. J. Denning. Educating a new engineer. *Communications of the ACM*, 35(12):83–97, 1992.

[6] P. J. Denning. CNE: Center for the new engineer. George Mason University, School of Information Technology and Engineering, Fairfax, VA, August 1993. Electronically Distributed Report.

[7] P. J. Denning. Designing new principles to sustain research in our universities: An open letter. *Communications of the ACM*, 36(7):99–104, 1993.

[8] S. Hill and T. Hill. *The collaborative classroom.* Heinemann, Portsmouth, NH, 1990.

[9] P. Kotler. *Marketing management: Analysis, planning, implementation, and control.* Prentice Hall, Englewood Cliffs, NJ, 1991.

[10] K. J. Maly, D. E. Ray, J. C. Wild, I. B. Levinstein, S. Olariu, C. M. Overstreet, N. S. V. Rao, T. Ireland, and G. Kantsios. Computer productivity initiative. In *Proceedings of the 7th SEI*

Conference on Software Engineering, page (to appear), Pittsburgh, PA, 1994. Software Engineering Institute.
[11] C. H. Marbury, F. S. Barnes, L. Lawsine, and N. C. Nicholson. A one room schoolhouse plan for engineering education. *IEEE Transactions on Education*, 34:303–309, 1991.
[12] G. M. McCue. IBM's Santa Teresa laboratory — Architectural design for program development. *IBM Systems Journal*, 17:4–25, 1978.
[13] G. J. Myers. *The Art of Software Testing*. John Wiley and Sons, New York, NY, 1979.
[14] P. Rosati. Student retention from first-year engineering related to personality type. In *IEEE and ASEE Frontiers in Education*, pages 37–37, Washington, D.C., 1993. IEEE Press.
[15] J. C. Schlimmer. A team-oriented approach to software training. In *IEEE and ASEE Frontiers in Education*, pages 308–312, Washington, D.C., 1993. IEEE Press.
[16] R. E. Slavin. *Cooperative Learning*. Longman, Inc., New York, NY, 1983.
[17] T. Stanton and K. Ali. *The Experienced Hand: A Student Manual for Making the Most of an Internship*. The Carroll Press, Cranston, RI, 1982.
[18] M. Weinstein and J. Goodman. *Playfair: Everybody's Guide to Noncompetitive Play*. Impact Publishers, San Luis Obispo, CA, 1980.
[19] W. A. Wulf. Development of a set of closed laboratories for an undergraduate computer science curriculum. Technical Report TR–91–17, Department of Computer Science, University of Virginia, Charlottesville, VA, 1991.

Appendix A. Team Selection and Evaluation Form[1]

To set personal development goals and for initial evaluation, in the first column, rank each applicable item as follows:

-2 Poor
-1 Below average
1 Average
+1 Very good
+2 Outstanding

To evaluate growth, in the second column, rank each applicable item as follows:

-2 Became worse
-1 Slacked off some
1 Stayed the same
+1 Some progress
+2 Greatly improved
Student struggled with this issue

Skills in Task Performance

___ ___ Completes assigned tasks
___ ___ Attends to detail
___ ___ Manages time and energy well
___ ___ Meets deadlines
___ ___ Understands and follows instructions
___ ___ Shows judgement about when to seek further guidance, when to be self-reliant
___ ___ Demonstrates the ability to organize thoughts; presents them clearly verbally or in writing
___ ___ Comfortably familiar with common algorithms and different programming styles

Work Habits and Presentation of Self

___ ___ Is punctual and dependable
___ ___ Conforms to expected team norms
___ ___ Is self-reliant (as appropriate)
___ ___ Looks for new responsibilities, takes initiative
___ ___ Dresses neatly and appropriately
___ ___ Has a pleasant, positive demeanor; appears confident, informed, and attentive to others

Attitudes

___ ___ Demonstrates active desire to learn from and contribute to team
___ ___ Has open mind; does not rush to judgement
___ ___ Accepts and makes positive use of criticism
___ ___ Understands and accepts necessity of some dull or repetitive tasks
___ ___ Demonstrates problem solving orientation; looks for positives in difficult situations; looks upon problems as challenging
___ ___ Is inquisitive
___ ___ Has respect for other people's different skills and life experiences
___ ___ Recognizes and accepts own limitations
___ ___ Willing to attempt new challenges
___ ___ Understands difference and strikes balance between roles of student/team member and between own goals/team goals
___ ___ Is cooperative, flexible, and adaptive
___ ___ Demonstrates ability to set and refine, then fulfill personal goals
___ ___ Shows openness to self-evaluation
___ ___ Seeks out resources within team, team director, and industry affiliates

Skills in Human Relations

___ ___ Adjusts to a variety of new circumstances, expectations, and people
___ ___ Develops new, alternative ways to respond especially when prior expectations are not met
___ ___ Shows ability to question and explore team/industrial affiliate organization, its methods, etc., without putting people on the defensive
___ ___ Is sensitive to the needs of others
___ ___ Is a good listener, attentive
___ ___ Copes well with unexpected problems
___ ___ Demonstrates tact
___ ___ Asserts own views and concerns effectively
___ ___ Has tolerance for ambiguity

[1] Adapted from [17]

Appendix B. Practicum Reading List

Semester 1 (Fall 1992)

What color is your parachute?, R. Bolles (1992). Ten Speed, Berkeley, CA. $13.

Marketing management: Analysis, planning, implementation, and control, P. Kotler (1991). Prentice Hall, Englewood Cliffs, NJ. $55.

PC Week. Ziff-Davis, New York, NY.

Semester 2 (Spring 1993)

The mythical man month: Essays on software engineering (1st Ed.), F. Brooks, Jr. (1975). Addison-Wesley, Reading, MA. $22.

The art of software testing, G. Myers (1979). John Wiley & Sons, New York, NY. $35.

PC Week. Ziff-Davis, New York, NY.

Semester 3 (Fall 1993)

Peopleware: Productive projects and teams, T. Demarco & T. Lister (1987). Dorset House, New York, NY. $25.

Project management for engineers, M. Rosenau, Jr. (1984). Van Nostrand Reinhold, New York, NY. $50.

PC Techniques. Scottsdale, AZ.

Semester 4 (Proposed)

Accidental empires: How the boys of silicon valley make their millions, battle foreign competition, and still can't get a date, R. Cringely (1992). Addison-Wesley, Reading, MA.

The one-minute manager: The quickest way to increase your own prosperity, K. Blanchard & S. Johnson (1981). Berkeley, New York, NY. $8.

Help! The art of computer technical support, R. Wilson (1992). Peachpit Press. $20.

PC Techniques. Scottsdale, AZ.

Semester 5 (Proposed)

Code complete: A practical handbook of software construction, S. McConnell (1993). Microsoft Press, Redmond, WA. $35.

PC Techniques. Scottsdale, AZ.

Semester 6 (Proposed)

Literate programming, D. E. Knuth (1992). University of Chicago Press, Chicago, IL. $25.

PC Techniques. Scottsdale, AZ.

Semester 7 (Proposed)

Programming pearls, J. Bentley (1986). Addison-Wesley, Reading, MA. $18.

PC Techniques. Scottsdale, AZ.

Semester 8 (Proposed)

More programming pearls: Confessions of a coder, J. Bentley (1988). Addison-Wesley, Reading, MA. $18.

PC Techniques. Scottsdale, AZ.

Jeffrey C. Schlimmer Dr. Schlimmer is an assistant professor of Computer Science at Washington State University. He received the Ph.D. from the University of California at Irvine in 1987. He is currently a member of the editorial board in his field of research, and is the author of several papers in the area of machine learning. He has served as program chair and reviewer for a number of artificial intelligence and machine learning conferences. Dr. Schlimmer is a member of AAAI, the Cognitive Science Society, and ASEE. His primary research interests lie in the areas of self-modeling databases and intelligent form-filling systems.

Justin B. Fletcher Currently a Ph.D candidate in computer science at Washington State University, Mr. Fletcher received the M.S. in Computer Science from the University of Idaho in 1983. His service to industry includes software development, management and consulting. Mr. Fletcher lectures on the C programming language and introductory computer science. His research focuses on hybrid symbolic-neural computational frameworks.

Leonard A. Hermens Mr. Hermens is a Ph.D. student in Computer Science at Washington State University. He received the M.S. in Software Engineering from the University of Idaho in 1991. He currently lectures in Software Engineering. Mr. Hermens is pursuing research in self-customizing applications using machine learning techniques. He is a member of IEEE, ACM, and AAAI.