Collaboration Across the Curriculum: A Disciplined Approach to Developing Team Skills

Ben Coleman
Moravian College
1200 Main Street
Bethlehem, PA 18018
coleman@cs.moravian.edu

Matthew Lang
Moravian College
1200 Main Street
Bethlehem, PA 18018
lang@cs.moravian.edu

ABSTRACT

Increasing the communication and collaborative skills of computer science students has been a priority in the community for some time. We present our philosophy, collaboration across the curriculum, which moves beyond existing individual courses or course units to teach collaboration skills in a pervasive manner. In our approach, concepts are introduced and skills are developed throughout the computer science curriculum—from CS1 to a capstone experience. Students are provided with opportunities to exercise skills in reflective environments that eventually mirror real-world experiences, and technical course content is not compromised.

We argue for this system and provide details about how collaboration across the curriculum is accomplished at a small liberal arts college.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education

General Terms
Human Factors

Keywords
Collaboration, Teamwork, Curriculum, Communication

1. INTRODUCTION

Whether the message comes from the corporate world [1, 24] or our colleagues in academia [8, 25], there is a demand for students with better communication and teams skills. The general goal of teaching communication skills is not unique to computer science; many schools have embraced Communication Across the Curriculum (CXC) or Communication in Disciplines (CID) [5, 20] as requirements for all students. Both programs develop interpersonal communication and public speaking, and though they differ in their implementations, the emphasis of both programs is on teaching communication skills throughout multiple courses.

Although these programs do not directly address collaboration skills, the strength of CXC and CID is their deliberate approach to skill development across the curriculum. Both programs emphasize a progressive approach to developing students’ abilities where later courses build upon ideas learned in earlier courses.

Since collaboration encompasses such a wide range of skills—from interpersonal communication and conflict resolution to team planning and division of labor—it is not possible for students to master collaboration in a single course. For students to truly learn to work in teams, collaboration needs to be considered a curricular goal, and our effort should be to determine appropriate places in the curriculum to introduce new collaborative skills.

We propose collaboration across the curriculum as an intentional approach to teaching team skills to students. In this system students are introduced to collaboration concepts at points in the curriculum that match their stage of development. Students in the introductory curriculum learn basic communication and collaboration skills in pairs. Based on these skills, students at the sophomore and junior level are assigned carefully planned-out experiences in group settings. Finally, in the senior capstone course students are expected to synthesize their previous experiences as they collaborate on a large-scale, semester-long project.

Throughout the curriculum, collaboration is taught following Kolb’s [12] model of learning by providing students opportunities to experience, reflect upon, hypothesize about, and plan their collaborations. Through repetition across the curriculum that builds upon prior experience, students have time to absorb these collaboration concepts and develop their skills.

Because our approach spreads this material across the curriculum, it allows collaboration to be taught in a way that supports individual course content rather than replacing it. Through simple restructuring of in-class activities, laboratory activities, and programming assignments, students learn to collaborate as they learn course content.

The community has generated significant literature about teaching collaboration and teamwork. We recognize its importance in Section 2 and recommend that this work be used in the system we propose. Section 3 presents our philosophy for teaching collaboration across the curriculum and Section 4 describes how we implemented this framework within our computer science major. We conclude by discussing the
results of our approach in Section 5.

2. RELATED WORK

We propose that students should learn collaboration skills through multiple experiences across the entire curriculum. There is a large body of work that documents the design of individual experiences (e.g., courses, workshops, etc.) and effective instruction for specific skills related to collaboration, and it is our intention to utilize this work in our curriculum-wide approach.

Within the computing disciplines, various groups have developed resources to support teaching the communication skills that form the basis of collaborative skills. For example, Carter et al. [4] have developed model curriculum and syllabi to teach communication skills in Software Engineering courses. Similarly, Smith et al. [32] describe a set of activities to help teach students soft skills such controlling and using emotion and employing appropriate language in business communications. Blume et al. [16] provide a collection of workshops to help students communicate with team members more effectively. Finally, Norback et al. [27] describe a web-based system to teach communication skills.

A variety of authors have written about techniques to support teams in individual courses. Although the work of Oakley et al. [23] was originally described in terms of cooperative learning, their ideas work equally well for traditional teams. In their paper, they provide a thorough discussion of team concepts and issues as well as a collection of materials to assist faculty. Similarly, Brown and Dobbie [3] discuss a tutorial to help students learn team skills. A variety of authors have written about aspects of forming teams and characteristics that affect team success [6, 17, 28]. Kurkovsky [13] describes four roles the instructor should role play during a group project. Finally, Smarkusky and Smith [30] describe an approach to teaching collaboration skills where communication skills are included as learning objectives in CS1/2 to prepare students for group work in later classes.

Instructors interested in utilizing a formal process for team interactions have a few options. “The TEAM Handbook” [26] describes a commonly-used system within corporations. In an academic context, Hilburn [9] discusses how to use the “Team Software Process” originally described by Humphrey [10]. Others, including Smarkusky et al. [29, 30] provide collections of modules or workshops to help students learn group skills.

Techniques to evaluate teams are an important area of research. Smith and Smarkusky [31] and Smarkusky et al. [29] provide a thorough coverage of evaluation principles and supply a variety of matrices to evaluate groups. Brown and Dobbie [3] discuss ways to directly and indirectly assess a team. Stone and Madigan [33] discuss the use of reflective writing to teach collaboration and provide a useful rubric for grading.


Extreme programming and the Agile process are popular paradigms in introductory courses. McKinney and Denton [18] use pair programming in the introductory sequence, and they discuss guidelines for making this technique effective as well as ways to assess students. Teles and Tolla de Oliveira [34] describe efforts to include communication skills across the curriculum and in a course on Agile development.

2.1 Collaborative Learning Pedagogies

Many popular alternatives to lecture-driven instruction use team-based systems to improve student learning. When these systems are used, students learn collaboration skills as a consequence of the team-based environment. However, this learning is a side-effect of the environment rather than a goal. Still, the evidence for their effectiveness in student learning makes them appealing, and they can be used within our proposed model.

In the Problem-Based Learning (PBL) model of learning, students work in teams to solve problems during class sessions. Newman and Faulkner [22] describe a variation called Open-Ended Group Projects where the problems collectively form a large software project.

In Peer-Led Team Learning (PLTL) small groups of students learn along side a student peer that recently completed the course. Murphy et al. [21] and Biggers et al. [2] describe lessons learned using this approach.

Popular in science classrooms, Process-Oriented Guided Inquiry Learning (POGIL) is a variation of PBL. The difference is that POGIL activities are more highly scripted; students construct an understanding of the material by answering questions that guide them through the learning cycle. Recently, Kussmaul [14] has developed activities that introduce POGIL in computer science classrooms.

For a comparison of these techniques, see the survey by Eberlein et al. [7].

3. COLLABORATION ACROSS THE CURRICULUM

Like writing or speaking well, collaboration must be developed throughout a student’s undergraduate career as a department effort. As educators, we must be intentional about how and when we introduce, develop, and expect mastery by our students of the relevant skills. Therefore, we propose a curriculum-wide approach to developing collaboration skills: collaboration across the curriculum.

We describe our philosophy at different stages in the curriculum. In required courses, we introduce concepts of collaboration at appropriate places and then build upon these experiences later in the curriculum—both in required and elective courses. Consequently, students have the opportunity to see that the development of collaboration skills is part of the natural progression through the major—much like their ability to design and implement efficient algorithms or utilize abstraction to build complex systems.

Along the way, we carefully balance individual learning of course content with collaborative experiences by recognizing that skills can be learned without group experiences. Finally, whenever possible we assess and evaluate the student’s reflection and analysis of a collaboration rather than the product of the collaboration.

At all levels of the curriculum, collaborative elements share a foundation in Kolb’s model [12]: as students are exposed to new ideas, they are given opportunities to practice the skills, and are regularly asked to reflect upon their performance and plan for future experiences.

Effective collaboration requires communication, interpersonal, and conflict-resolution skills. Beyond these elements, students must learn individual abilities—recognizing their own strengths and weaknesses, trusting in the competence of others, committing themselves to team goals, and accom-
modulating differing personal values and ideas—and subsequently learn to use these tools to create team environments in which members act in a coordinated manner, communicate with one another, share decisions, resolve conflicts, and produce collectively-built products [11, 19].

The development of collaboration begins by laying foundations in CS1 and CS2. However, in the introductory curriculum it is also important that students individually learn foundational principles. Combined with the fact that these courses already contain numerous requirements, the inclusion of collaboration skills presents a challenge. To balance these competing needs, we believe the introductory curriculum should teach basic communication skills and have multiple small-scale interactions. We recommend that these interactions be structured so that they meet the following guidelines:

- **Supervised assignments** either assigned in class or in a laboratory give teams mutual accountability.
- Assignments that revolve around a single concept provide student teams with a shared understanding of their task.
- **Pair-programming** limits the complexity of the team mechanics and results in a collectively-produced product.
- Time is set aside for reflective discussion of team experiences, so that elements that drive team success are highlighted.

Students that succeed in the introductory level have demonstrated a competency with basic programming techniques and are better prepared for traditional team-based programming. As they continue through the curriculum, we should build upon their success to further develop their collaboration skills. The focus of this development should be in required courses, as this guarantees that all students see the material. However, it is also important that students gain experience with collaboration in elective courses. Thus, our efforts should be tailored to the course and its place in the major.

Regardless of the course, students at this level are not necessarily ready for truly independent team work. When given open-ended projects with loosely-defined deliverable products, students tend to revert to a “group work” mentality: project elements are divided—most likely unequally—among group members who work independently and the final product of the group is simply the collection of their individual products.

Experiences at this level should be highly-structured but allow students to apply the skills learned in the introductory courses:

- **Clearly-defined specifications** provide team members with a shared understanding of the team’s product and **objective rubrics** provide students with a shared understanding of success.
- Assignments should provide opportunities for real collaboration, but be **limited in duration** so that teams can incorporate feedback in future assignments.
- **Individual work products** like write-ups or experience reports allow instructors to assess individual skills and provide students with opportunities for honest peer-review.

As students gain experience, the quantity of scaffolding can be reduced and the size of projects can increase. The key point is that by planning how the experience changes throughout the curriculum, students have the opportunity to build upon success and truly learn collaborative skills.

After multiple such experiences, students are prepared to explore higher-level frameworks and paradigms of collaboration in a capstone course. In this course their prior experience allows them to appreciate how new techniques contribute to a team process.

### 3.1 A Deliberate Approach

Though none of the elements described above are necessarily novel, the program-wide decision to purposefully and deliberately introduce and develop collaborative skills **across the curriculum** is not one that appears in the literature.

If, as a community, we believe that collaborative skills are an essential element to a computer science education, we need to intentionally structure our course activities to support the development of this skill, that—like technical topics—courses should be linked in their development of this skill, and that departments and programs should decide as a whole how collaborative skills should be emphasized at different points in their curricula.

The approaches described in Section 2 will certainly be a part of these efforts; they have proven valuable enough to be supported by and of interest to the community. In fact, some of the ideas presented by others are part of the implementation of our philosophy.

### 4. OUR IMPLEMENTATION

We implemented a collaboration across the curriculum program at a small liberal arts college. Because of a limited variety of programs within the college, the development of most “soft skills” came from within the computer science program. Additionally, the college uses a unit system where a major contains twelve courses. Three of these courses (Calculus I and II and Discrete Mathematics) already lie outside of the computer science program. As a result, we could not require students to take courses that specifically address additional objectives, nor could we relax the technical objectives of existing courses.

Fortunately, the effects of these constraints aligned with our philosophy. Our implementation follows the model we propose, builds on the work presented in Section 2, and, as described in Section 5, has been successful in building collaboration skills in our students.

#### 4.1 Introductory Courses

The process of developing collaborative skills begins immediately in the introductory CS1/CS2 sequence. However, before we can actively engage students in collaborative work, we need to break them out of thinking that collaboration is unethical—an idea many incoming freshmen learned in their high schools. In the introductory courses we encourage the students to work together on all assignments except tests, and we spend time in class helping them understand what responsible collaboration means. In addition, we actively practice paired-programming in our closed labs—again with instruction about how to work effectively with a partner. Combined with frequent in-class activities using active, group-based learning, students come to see the value of collaboration.
To discourage leeching, tests and the final are worth 40% of the overall grade; it is difficult to do well in the course without learning the material. The active nature of the class sessions also makes it obvious which students are coasting.

Closed laboratory activities are structured like POGIL [14] activities. Working in pairs, students are presented with a problem of the week, typically based on a simple game. Solving the problem is done by writing a small program to simulate the game and collect data. Before they begin, each pair explores the problem by hand and predicts the final outcome. After they collect data, they comment on their results and their original prediction.

Each week, classroom time is set aside as a weekly review session. During these sessions, students are asked to reflect upon positive and negative aspects of working in groups and plan how to leverage successes and avoid failures in future collaborative experiences.

The strength of this approach is that in-class activities, closed-labs, and homework assignments provide multiple collaborative experiences for students. Through in-class discussions and reflective writings, we prompt the students to reflect on their experiences, hypothesize alternative approaches, and plan for future collaborations. Students are then able to test their plans in next week’s lab, tomorrow’s class, or on the homework due Wednesday. This gives students an opportunity for multiple iterations through the learning cycle and to begin learning how to collaborate effectively.

4.2 Core Curriculum

During the second year, students have three additional courses they are required to take, two in the first semester and one in the second. These courses allow us to plan experiences that build upon the skills learned in CS1/CS2. In elective courses, experiences must be crafted to allow for students with differing levels of exposure to working in teams.

In the first two required courses, Algorithms and Data Structures and Computer Organization, students are given a series of team-based programming projects. Prior to the assignment of the first project, students learn about Tuckman’s [35] theory of groups. After each project, students are asked to reflect about their group experience and relate it to Tuckman’s theory. Since these courses are not about designing software, the specifications for these projects are very detailed. Further, the projects are short enough to reduce the complexity of the collaborative effort.

Students are graded on individually-produced write-ups that describe the process of creating the program as well as their team experience. Like CS1 and CS2, in-class reflection of team experiences follows project deadlines and students are given individual and collective feedback through their write-ups.

The third required course, an Introduction to Software Engineering, helps students begin the transition from these highly-structured projects to a more open-ended system where students must make design decisions, split up work, and handle the inevitable conflicts between team members.

The course is designed around a semester-long project with several regular checkpoints. Deliverables, including a proposal, various stages of design documents, and implementation, are due at points throughout the semester. Teams receive feedback related to the technical aspects of their project through these deliverables. Teams are also required to have regular informal meetings with the instructor. Though technical topics are discussed in these meetings, students are also asked to assess the functioning of their team and are given direction and feedback from the instructor. These regular meetings allow students to see how the dynamics and mechanics of their teams are within their control and provide them with tools necessary to function successfully on an open-ended project.

Although the intermediate deliverables are important, a large part of a student’s grade on this project is based on the analysis of the team experience. This prevents students from ignoring the team aspect of the project.

Students are also given a series of individual labs throughout the semester. These assignments develop and assess a student’s software engineering skills as well as provide students with individual control of their final grade in the course.

Together, these three required courses introduce students to the major themes of collaborative software development. Based on these experiences, students are prepared for a set of elective courses that, in addition to the content goals, include a department-mandated requirement to further the students’ development of collaboration skills. We provide two examples that illustrate the range of possibilities.

4.3 Elective Courses

Game Programming is a senior-level elective where student pairs implement a video game of their design. The learning objectives of the course focus on good software design, and each team must negotiate the design process for various aspects of their game. In-class discussions feature the relationship between good design and effective collaboration.

As the course progresses through various elements of game programming (e.g., coordinate systems, agent behavior models, efficiency issues, collision detection, graph algorithms, path planning, etc.), team projects are given corresponding checkpoints. Instructor meetings with each team at checkpoints help monitor the progress of each group’s collaboration. A group’s final assessment, while dominated by the quality of their product, includes a significant element where students evaluate the collaboration.

In Artificial Intelligence, students complete a traditional semester-long research project in teams of three or four. The course attracts students with a wide range of technical experience (from freshmen to seniors), and groups are assigned by the instructor to ensure that each group reflects this diversity. While the research projects themselves are open-ended, the progression of each project is carefully scripted. Regular assessment of team performance occurs via in-class discussion, regular team meetings with the instructor, and periodic peer-review.

4.4 Capstone Experience

After multiple experiences in a structured setting, our students are ready for a “real” team experience. During their last semester of study, students are required to take a course titled “System Design and Implementation.” As the capstone experience for students in the major, this course requires students to pull together ideas and skills from across the curriculum. Thus, it is perfectly positioned for the students to demonstrate mastery of the collaboration skills they have learned earlier in the curriculum and allow them to
practice further team skills.

To further their theoretical understanding of software development via collaboration, students learn about a variety of software development frameworks and practices. Our goal is to expose students to a breadth of techniques and have them think about how the techniques contribute to software development. Students are assigned a variety of readings on advanced software methodologies such as refactoring, code review, and version control, and they prepare short reflective journal entries on each reading. These writing assignments form a basis for discussion in class.

As the students are learning these practices, they are introduced to the collaborative project for the course. Our goal is to provide the students with experience working on a software project with real clients and customers. In its current offering, students are developing education-related software for an NSF-funded research consortium. Through this project, students are exposed to all aspects of the software process—from requirements elicitation to testing. All phases of development are completed by teams of students.

Because our grading class sizes are small (typically five to ten), students participate on multiple teams simultaneously. Through these teams, the students experience different aspects of team development because they are assigned roles in the teams based on their strengths. The instructor for the course assigns students to tasks, and by the end of the semester, each student is the team lead for at least one task.

By mixing conceptual material from the texts and hands-on tasks for the project, students gain a deeper appreciation for the content of the course. Students may appreciate the ideas behind version control patterns, but they truly understand them after they resolve conflicts or inadvertently break the build. In many cases, students try different approaches to collaborative work simply because the discussion of content naturally includes conversation about how to apply the material to current tasks. In other situations, students are required to apply new skills. For example, regular code reviews are mandated by the instructor, and the evaluation frequently results in refactoring assignments for one or more teams.

Because the primary learning objective for the course is the collaborative process of creating software, students are never graded on whether tasks are completed successfully. Instead, grades are based on self-evaluation and peer review. The final paper for the course asks the students to reflect on their experience with the project. By acknowledging various successes and failures during the semester, students gain context for the material of the course as well as their contribution to collaborative development. Along with this reflective paper, students submit a log of their efforts on the project during the semester as well as a peer evaluation of the contributions of the other students.

5. RESULTS AND DISCUSSION

Measuring whether students learn to collaborate is difficult, and at a school with only five to ten graduates per year, size confounds assessment. However, the principles used to guide the implementation of this curricular objective are informed by research both within and outside the CS education community. Further, anecdotal evidence indicates that our approach is working. Since we implemented this system, the students in the senior capstone are more successful at team work. They are better prepared for the day-to-day interactions with peers, allowing the course to cover more advanced topics in software engineering. In a reflective writing assignment at the end of the semester, the students identify team skills as a useful product of the course. In addition, recent graduates indicate they are better prepared for collaborative efforts in the workplace than their peers from other institutions:

- One student reports that his collaborative experiences in college were instrumental in obtaining a job, saying, “teammates were what they look for most of all, and [SCHOOL NAME]’s technique obviously facilitated growth in that direction.”
- Another student reports that utilizing pair-programming and code-review allowed him to integrate quickly into the team while a peer from another school struggled to get beyond a “solo programmer” approach.
- Another student reports that not only is his supervisor impressed with the refactoring he performs, but that he has helped other, more senior developers adopt these techniques.

For students to truly understand how to work on a team, they need to learn and practice collaboration in multiple courses, similar to how CXC/CID teach communication skills or Writing Across the Curriculum programs teach writing. We must think intentionally about appropriate places in the curriculum to introduce collaboration skills and recognize that, particularly at the introductory level, these skills are best learned using activities other than traditional group programming experiences. Regardless of the actual activity, students should be forced to reflect upon their experience, and faculty should utilize both peer-evaluation and reflective writing to evaluate each student’s performance in the group rather than the final product. Finally, the actual use of group work should be crafted to support the students’ stage of development with collaboration. Students are ready for open-ended projects at the senior level, but until then scaffolding should be in place.

We described our approach, but each school needs to shape collaboration across the curriculum to fit their program.

6. REFERENCES


