Integrating Communication Skills in Data Structures and Algorithms Courses

William Eberle
Tennessee Tech
Cookeville, TN USA

John Karro
Miami University
Miami, OH USA

Neal Lerner
Northeastern University
Boston, MA USA

Matthias Stallmann
NC State University
Raleigh, NC USA

***Abstract*—While the improvement of computer science students' communication skills has frequently been called for in the literature, employers continue to feel that recent graduates are not equipped with the writing, speaking, and teaming skills essential in the 21st century workplace. One problem with previous approaches is that they often teach communication skills in dedicated courses rather than integrating them into technical classes across the curriculum. In this paper, we report on a multi-institutional faculty team's efforts to integrate communication skills into mid-level data structures and algorithms courses as part of a larger NSF-funded project to enact integrated reform throughout computer science/software engineering curricula. We present an outline of assignments designed to develop communication skills (writing, speaking, reading, listening, and teaming) intertwined with technical skills, and discuss our preliminary efforts to assess these efforts. Our work reflects a general approach to incorporate communication activities *within the computer science curricula* and to help students learn and communicate technical content in academic and professional settings.**

Keywords— Data Structures, Algorithms, Communication Skills.

I. Introduction

Based upon years of experience in industry, coupled with direct contact with students in academia, we can emphatically state the obvious: most computer science students do not know how to communicate well. The technical skills new employees possess are potentially lost due to an inability to effectively articulate a coherent design, idea, or plan [1]. Even students who land their dream job often have difficulty working as part of a team or presenting ideas, either orally or in writing, to peers, management, or customers. These difficulties lead to frustration and an inability to make a difference in their field.

For many computer science students, communication curricula are part of their general liberal arts education and are typically focused on the social sciences and humanities. As a result, students are often not learning and practicing relevant communication skills (writing, speaking, and teaming) *in their discipline*, and often do not see the importance of developing those skills. Frequently they imagine the typical computer science job as sitting in an isolated cubicle, trying to build the latest video game or killer app.

As part of a National Science Foundation CPATH grant, we teamed up with academics and professionals, including communication-across-the-curriculum specialists from the United States and abroad, to address this gap [2]. By having a consistent thread of communication instruction for students throughout their computer science education, ranging from introductory Computer Science to Software Engineering and Capstone courses, the emphasis on integrating communication skills (reading, writing, listening, speaking, and teaming) with technical content becomes a primary curricular feature. With this approach, students not only acquire the technical and analytical capabilities they need, but are also able to communicate effectively and excel in their field.

This paper focuses on our work with integrating communication skills into *data structures and algorithms* courses, early in the curricula (typically after CS1). Two years were spent designing assignments to expose students to both technical concepts learned in class and the related communications skills necessary to provide coherent, complete, and professional deliverables. Faculty members from four institutions, including a communication specialist, were involved in creating and implementing this new curriculum, evaluating its effectiveness, and refining assignments to make them available to colleagues at other institutions.

Here we offer related work in Section II, and describe the participating institutions in Section III. Section IV presents the different assignment paradigms that were implemented to address different communication skills. Finally, Section V presents some reflections, preliminary evaluations, concluding thoughts and potential future directions.

II. Related Work

A number of universities have attempted to create courses for their majors that deal exclusively with communication skills. Northwest Missouri State University addresses the issue of oral communication skills for their computer science undergraduates through a seminar course [3]. Denison University introduces communication skills to computer science and mathematics students through a jointly led lab [4], focusing on the improvement of their oral communication skills. The University of Toronto created a new course entitled "Communication Skills for Computer Scientists" [5]. Still, as noted previously, rather than integrating communication skills within existing computer science courses, these institutions focus on the creation of a separate course that works solely on students' writing, speaking, and interpersonal communications.

More relevant to our project, Hartman introduced writing skills into a data structures course, specifically in assignments dealing with analysis of algorithms [6]. Based on accreditation guidelines, Beard et al. identified the soft-skills most sought by

FIE'13, October 23-26, 2013, Oklahoma City, Oklahoma, USA.

employers, and created a model for producing and evaluating relevant activities [7]. However, the curriculum was designed around accounting courses and not traditional computer science. Falkner et al. present a theoretical framework for assisting instructors with integrating communication skills in the introductory computer science courses (i.e., CS1/CS2) [8]. They provide some guiding principles and methodologies that can be incorporated early in the computer science curricula, along with initial feedback from students.

A broad view of communication skills, including writing, speaking, and teamwork, has also been supported in several studies. Gruba and Sondergaard report on the use of a conference run by the students in a computer science course so that they can work on their communication skills as well as learn technical content[9]. Students were tasked with the responsibility to create, host, and participate in a public research conference, offering opportunities for a wide range of communication activities in a real-world setting. In a 2006 paper, Hoffman et al. describe activities at Quinnipiac University to capitalize on the potential for communication tasks to help students "write to learn" technical content, as well as to communicate that learning [10].

A caution on curricular redesign is provided by Cilleirs, who found a discrepancy between instructor and student perceptions of the value of communication, particularly writing, activities [11]. The study suggests that while students perceive academic writing activities as beneficial in the construction of a report, many times the actual activities used by instructors are not perceived by students as being useful. Such cautions are essential to consider when integrating communication assignments to fulfill technical content learning.

III. INSTITUTIONAL CONTEXTS

In what follows, we describe each of our institutional contexts: overall demographics, the computer science curriculum, and the place of data structures and algorithms within that curriculum.

Miami University
Miami University, located in Oxford, OH, is a mid-sized public University stressing a balance between research and teaching. The Department of Computer Science and Software Engineering has about 300 undergraduate majors and 20 Masters' students, with a typical class size from 20 to 40. The data structures course is the third programming course, while the algorithms course is generally taken by upper classmen.

North Carolina State University
North Carolina State University, located in Raleigh, NC, is a large, public research-oriented institution. The computer science department has about 600 undergraduate majors and roughly the same number of graduate students. Except for the CS1 course, a multi-section lecture/lab combination with 30 students per section, a typical core undergraduate class has from 60 to over 100 students. Class size presents a special challenge to an instructor who wants to introduce communication skills. As at Miami, the data structures course is the third in a sequence of Java-based programming courses.

Tennessee Technological University
Tennessee Technological University (TTU), located in Cookeville, TN, is classified as a medium-sized, public, rural university, with a computer science enrollment of over 300 students, primarily undergraduates. Most students come into the program with little programming experience, having had no access to computer science classes in high school. At TTU, the data structures and algorithms course is second in the introductory sequence of C++ courses, with class size ranging from 40 to 70 students.

IV. ASSIGNMENT PARADIGMS FOR COMMUNICATION SKILLS

This project defines *communication* in terms of five modes: reading, writing, listening, speaking, and teaming. While data structures and algorithms courses feature considerable technical content, at the core they require students to design solutions (specific data structures or algorithmic approaches) to fit particular computing problems. This design element and the implementation of the design quite naturally lead to a wide variety of communication activities that mirror the types of communication students will need to do in the workplace; thus, the real-world element adds value to these tasks and is often highly motivating for students.

In the following sections we discuss various paradigms or overarching categories that we utilized to incorporate communication skills within a data structures and algorithms course. For each paradigm, we present a brief description of the work-place or professional scenario for the particular communication skills, as well as offer assignment examples. One will note that many of the concepts we implemented are applicable to courses other than data structures.

All of the assignment frameworks described in this paper and detailed descriptions of the assignments (as well as others) can be found at http://cs-comm.lib.muohio.edu/.

A. Program Design: Reading and Writing

Designing and implementing a program is probably the most common assignment (across all institutions) for students. They are given a problem to be solved, and they are responsible for designing, building and testing their solution. With this type of assignment, we incorporate two communication skills: reading and writing.

The *reading* skill is manifested in giving the students a task that requires them to do research. For example, in one assignment, we require students to implement a random number generator, a topic not covered in their textbook, and thus requires they search the literature – a situation they will encounter many times in the workplace. The particular goal is not important; the assignment template is constructed in such a way that the the appropriate communication skills are independent of the data structures concepts being introduced (e.g., stacks or queues) and thus can easily be adapted by other instructors to their own assignments.

The *writing* skill is integrated into the assignment by having the students design an application from scratch. To provide some guidance, a design template is provided that includes three parts for the students to populate: pseudo-code;

design decisions; and design issues/notes. The students limit their design document to no more than two pages (relieving some of the grading workload), and are required to submit coherent, grammatically correct, well-written text. A key concept the students must understand is that in the workplace, an employee's ability to convey their ideas in written form is important at many levels: conveying design decisions to their peers, presenting ideas to project managers, and presenting potentially transformative ideas to upper management.

B. *Omitting Details: Listening and Writing*

In the workplace, not only must software engineers be able to read technical material and write coherently, but they also must listen well. Customers will sit down with the problem solvers to talk about their problem(s), and it is up to the developer to listen and ask appropriate questions.

In the *listening skills* paradigm, assignments are handed out in paper form or posted on the class website, where class time is spent reviewing the assignment and answering questions. To integrate listening exercises, the assignment explanation includes two *verbal requirements* that cannot be found in the written description – the discussion of a feature that the program needs to accomplish is omitted. Because the students are told upfront that some of the required features of the assignment will be given verbally, and cannot be found anywhere in the written documentation that was distributed, the students listen carefully to the program specifications and ask questions about what they just heard.

C. *Collaborative Design: Teaming, Reading, Writing, Speaking and Listening*

The ability to work in a team is arguably the most important communication skill. Most software engineering projects involve multiple people, from business analysts to designers, coders, testers, managers, and customers. The ability to communicate both internally (i.e., within your company) as well as externally (i.e., with your customer) is vital to the success of a project.

This paradigm can incorporate all five of the communication skills: reading, writing, teaming, speaking, and listening. The reading and writing skills occur through the same avenues as mentioned previously. However, because the initial design is done as a team, students can view and analyze other students' interpretations of the assignment (i.e., what they read), as well as see and comprehend other students' design ideas (i.e., through their writings).

We present two examples of the collaborative-design paradigm, used at two different universities. Both involve team effort toward writing a design and giving an (optional) presentation of it. Each student, working individually, is then required to implement a design, not necessarily the one proposed by their own team. These examples can be adapted to many practically-motivated situations in which the data structure(s) to be used are not explicitly given.

Example 1

The students explore the idea of using queues to create a simulator, such as an airport runway or a car wash.

The initial design is done as a team exercise, with some minimal time given during class for team meetings, followed by individual implementations of their chosen design. By working together, students will learn different approaches for creating a design from other members of their team. The first deliverable, an initial design document, is submitted for a grade about one week after receiving the assignment. This requires students to craft a design before implementing a program – combatting the common tendency to create a design as an afterthought.

After the teams have submitted their design document, one class period is spent on 5-7 minute design presentations. This exercise incorporates speaking and reinforces listening in a different way. To get a grade for their presentation, each student must present some aspect of their team's design. This is a light introduction to public speaking, as each student only gets about 1-2 minutes to talk. In addition, because students are standing in the front of the class with their team, it is less intimidating as they have a support group for answering any questions from the class. Listening is reinforced because students are permitted to use any design they want for their actual implementation, so by listening and paying attention to other teams' designs, they may find a design choice they prefer better than their own. While not a gradable aspect of the assignment, it is definitely motivating to the students to know that they can use another team's design.

After working as a team on an initial design document (and presentation), the students are then on their own to actually implement a solution. And, as mentioned above, they can choose to use their team's design, another team's design, or their own individual design.

Example 2

The students explore an approach to searching and replacing data in a text file as motivated by the following situation: Suppose that the only strings you are allowed to replace are words – contiguous strings with no embedded spaces or punctuation. A situation like this might occur if you want to rename variables in a program.

As in the previous example, there are two phases: a team design followed by individual implementation. Because this assignment is more complex – non-trivial interaction between data structures and more design decisions – the allotted time for the design document is three weeks, with an additional week for preparing the presentation. Team dynamics thus becomes a more prominent factor. An additional feature in this assignment is a test plan, submitted along with the design. In the workplace, developers and testers must often communicate about designs that will integrate a new feature (e.g., the word search/replace added to an editor). A test plan is a critical aspect of the communication between a requirements engineer and the designer. It ensures that each understands the expected behavior of the software under a variety of circumstances

D. *Justifying Choices: Reflecting and Writing*

As teachers, we hope our students learn from their mistakes and apply their knowledge to solving new problems. As we use exams to gauge students' understanding of course material, programming assignments are intended to see if they can apply what they have learned in the construction of a piece of software.

After students have crafted designs as members of a team and read (or heard presentations of) the designs submitted by others, their learning can be further enhanced if they are challenged to compose a design document individually. For this task to be successful, it is important that the instructor evaluate the design *before* an implementation (if any) is submitted. Students are, in effect, being asked to make judgments about their own work and that of others and to apply these judgments to their subsequent work. For instance, in our examples, students are successful when they understand the pros and cons of different data structures. Reflection on prior experience with communication skills, particularly writing, but also speaking, is essential for continued improvement of these skills in the workplace. Naturally the proposed paradigm is applicable in any context where students have had opportunity to interact with the work (writing or speaking) of their peers.

E. Experimental Comparison: Writing

The choice of an appropriate data structure or algorithm is often based on carefully crafted experiments using relevant problem instances. Students can be asked to apply competing algorithms (or data structures) to large instances of the same problem, drawn from both real and randomly generated data. In one example assignment, students are asked to compare six different algorithms for counting the number of occurrences of each word in a text. The assignment provides a collection of large text files, drawn from articles and books, and a generator for random text files with various characteristics. The writing component of this assignment is the creation of a report that outlines the scope of the experiment (algorithms and instances used), the results obtained and the interpretations thereof (e.g., does theoretical analysis predict actual run-time?). Creating useful and evocative charts – difficult even for experienced writers – is an important component of this type of assignment.

F. Creative Endeavors: Reading, Writing, Teaming and Speaking

In addition to the traditional classroom modes of gauging students' communication skills, students can demonstrate various communication skills through creative endeavors. Exercises include using media such as blogs and wikis to allow students to implement someone else's design (reading and comprehension) or communicate within their team (writing and teaming).

In one example, large class size makes it infeasible to assign individual (i.e., non-team) design assignments. Thus, a twist on the design presentations is employed using YouTube. Students are directed to post a video, limited to three minutes, containing a discussion of their design. Advantages of this approach include: (1) allowing students to work on their speaking skills in a non-intimidating environment; (2) prompting students to learn how to use a popular media web-site; and (3) allowing instructors to watch the design presentations at their leisure in a short amount of time.

G. Organization and Clarity Through Proofs: Writing

In one institution's theoretically-oriented algorithms course, we assigned problems as much for the writing challenge as well as the technical challenge. Correctness proofs and problem reductions tend to work well: such proofs frequently address a single concept, allowing the student to better focus on a well-written proof. The nature of many proofs dictate a natural template for students once they have solved the problem, and this structure simplifies the problem of providing feedback. It is easy to identify and comment on a failure to mention the crucial point, or to build steps of the proof in a logical and complete progression.

NP-Hardness reductions provide a particularly good example of how proofs can be effectively used to develop writing skills. The central element of such a proof is that of reducing one problem to another in polynomial time, i.e., to show that a polynomial-time algorithm for problem A implies one for problem B. To make this main idea more accessible to students we ask them to think of the reduction in terms of an implementation: we suppose the existence of a (black-box) implementation of an algorithm for A and ask them to design an algorithm for B using it. Given carefully chosen problem-pairs (Partition to Sum of Subsets being a good place to start), even the weaker students can usually grasp the idea.

The more challenging part of this exercise, one that involves communication skills, is for students to prove that the reduction is correct. They need to prove two different results (that the proposed transformation never results in either false positives or false negatives). Within each proof direction, the student will need to clearly state the central point and support it with arguments – not technically difficult, but challenging for the writer to lay out clearly. The writer must avoid conflating distinct concepts (leading to lack of clarity); leaving out one of the two arguments altogether; or failing to properly structure one or both of the arguments (e.g., neglecting to explain what they are proving). Providing clear, useful feedback is usually a simple task: the almost mandatory structure of the solution allows the grader to easily identify problems with the writing and explain why these problems inhibit clarity.

V. REFLECTION AND EVALUATION

While systematic evaluation of the success of our curricular and pedagogical efforts was not always possible, we offer in this section a combination of reflection, evaluation, and student impressions (from survey data) as a starting point for revision of our assignments. We structure this section around the particular communication modes and tasks we implemented.

A. Reflection and Writing Activities

Design and Experimental Comparison Documents. Design documents are pervasive throughout the curriculum and in the workplace. Data structures and algorithms courses often present the first situation where design focuses on more than the implementation of a simple algorithm or a single C++/Java class. Among the design documents submitted (for Example 2, Section IV.C), most lacked a good overview, consisting primarily of detailed pseudo-code and/or detailed UML. These students had learned UML in a previous course but had only limited exposure to high-level pseudo-code. Writing a well-organized overview is the most important, and apparently the most lacking, skill. One could use examples of excellent work from a previous semester – one student came up with a professional quality document – as models, provided that the assignments were not too similar. Models addressing the appropriate level of competence are harder to craft or find in the literature.

The experimental comparison assignment imposed a challenge not normally encountered in the curriculum, yet extremely important in the workplace: presenting data in the form of tables and charts with explanations. The primary issue with the students' charts was scaling: lines frequently ended up on top of each other. Tables were often hard to read; they did not line up properly or presented far too much detail. In explanations of the data there were two issues: misguided explanations (technical) and poorly organized blow-by-blow descriptions (writing). Future uses of similar assignments might be preceded by examples from the literature, where both good and bad examples of data presentation abound.

Proof-based Exercises. In the proof assignments there was a core set of writing-related problems that reflect specific problem types in the student's general writing ability. Examples include:

Failure to explain premise (poor writing structure): The average student frequently fails to explain what they are proving. Presenting a clear thesis is necessary in good writing; in identifying this problem we hope to help students organize their thoughts and lay a proper foundation for their writing.

Excessive use of notation (poor presentation): Students confuse "proof" with "algebra," apparently believing the latter is mandatory in the former. We emphasize that notation should be used only to facilitate understanding (as short-hand for concepts too cumbersome to write in prose): a proof without Greek symbols is acceptable. Making students aware of overuse of notation forces them to focus on clarity and on the use of appropriate tools.

Failure to connect ideas (poor logical presentation): Students often make logical jumps or fail to explain connections they have correctly made in their own minds. This problem becomes easier to both spot and explain in proofs than essay-based writing, allowing us to provide useful feedback.

Superfluous statements (problems with concise writing): It is not unusual for a student to attempt to say the right thing by way of saying everything. Again, in a mathematical proof it is generally easy to spot irrelevant comments and provide useful feedback.

The above observations about proofs are based on an upper-level algorithms class, but the ideas apply also to data structures classes that incorporate smaller proofs into project assignments. For example, students may be asked to prove assertions related to the correctness or runtime of a program.

B. Reflections on Speaking Activities

On the whole, both the formal in-class presentations and less-formal, student-made videos were successful in helping with speaking skills. For the collaborative design assignment most groups gave what appeared to be well-rehearsed presentations, given either by one group member or several. Almost all presentations adhered to the time limit (imposed in Example 2 of Section IV.C), allowing for a lively question and answer period.

We found that most of the students enjoyed the YouTube *speaking* exercise and were sometimes very creative in the process. Submissions ranged from voice-narrated computer animation to a "60 Minutes" television show parody. The ease of using video equipment, usually embedded in their laptops, made this a fun and easy assignment for the students and an effective vehicle for improving communication skills.

C. Reflections on Teaming Activities

In capstone courses, teams are often based on a set of complex criteria (e.g. Layton et al. [12]) but this approach is unnecessarily time-consuming for our purposes. In our case, team assignments were based on a ranking of students with respect to performance earlier in the semester, using one of two strategies to form groups: (1) including a range of student ranks in each group; or (2) grouping students by rank [13].

In using strategy (1), we hoped that weaker students would learn from stronger ones. This outcome was observed directly in a few cases. Of 18 teams of size three or four students, most appeared cohesive (based on student peer evaluations and instructor observation). However, three of the teams had one member who contributed little or nothing and another team had a member who was completely unable to contact the other three, and therefore ended up doing the assignment alone.

One of the authors had used strategy (1) in an earlier semester, but switched to strategy (2) based on the work of Braught et al. [13]. The justification is that a group of good students will be driven to produce even more, while a group of poor students will realize that they need to step up if they want to succeed. After making this change in team dynamics, the instructor noticed superior work from the top students – they reached out and tried interesting ideas – while most members of the bottom groups actually contributed to their team's efforts, sometimes with surprisingly good results. One disadvantage of the latter approach, observed in a context requiring more complex tasks, is that the poor students make little, if any, progress.

All communication skill assignments presented grading challenges. We relied solely on peer ratings for assessment of teamwork. Speaking was only lightly graded: any reasonable attempt resulted in credit. Thus, most of the grading burden focused on writing. One of the authors advertised a rough breakdown to the students (10 points for each of several aspects), but was then faced with the difficulty of deciding between, say, a 4 and a 7; furthermore, each such decision had to be justified. A more reasonable approach would be a simple checklist of items each worth only 1 or 2 points, such as "a table that summarizes data effectively." This would obviate the need for justification and allow the instructor to focus instead on constructive feedback: positive encouragement and suggestions for improvement.

D. *Students' Reflections on Communication Activities*

While logistics and a lack of resources precluded comprehensive assessment of this project, we did conduct an assessment of student perceptions of their communication skills before and after a team design assignment in one course – see Section IV-C. Students were given an attitude survey at the beginning and end of the semester; 32 out of 61 students participated. The questions asked students to rate their ability in each of the following: (a) *reading* technical specifications, including assignments, documentation, etc.; (b) *writing* technical documents, including descriptions of algorithms and experimental results, designs, etc.; (c) giving audience appropriate presentations (*speaking*); and (d) working effectively with a team of peers to accomplish a common goal (*teaming*). Ratings ranged from very good (5) to very poor (1).

On average, the students rated themselves more positively at the end of the semester in three of the four categories: the average scores increased from 4.09±0.13 to 4.25 ±0.13 for reading, 3.50±0.12 to 3.75±0.11 for writing and 3.66±0.14 to 3.78±0.14 for speaking. (The ±'s here indicate standard error.) Teaming was a different story: there the average rating dropped from 4.19±0.13 to 4.16±0.12. Students were more confident about their teaming ability at the beginning of the semester than about any other skill. The lack of improvement might reflect the fact that nine students (in the sample of 32), who rated themselves more poorly at the end, were predominantly ones whose teams fared badly. The presence of carefully designed and positively regarded team assignments in the second semester Java course and an emphasis on pair programming in the introductory course could explain the initial high confidence in teaming ability. Clearly more careful attention could be paid to the teaming aspect of our proposed assignments.

Prior experiences in these skills were elicited with the prompt, "Please list the courses (including CSC 316) and/or industry settings (e.g., co-ops) in which you practiced [reading, writing, speaking or teaming skills listed in detail]." Only 21 of the 32 students reported having done reading in CS1 even though the prompt specifically mentioned "assignment specifications." The number increased to 31 for CS2. Prior writing experience was reported by 5 for CS1 and 23 for CS2. Teaming went from 8 in CS1 (pair programming is a part of that course) to 27 in CS2, where teams of four or five are now standard. The significant prior (presumably positive) teaming experience explains the beginning-of-semester confidence (and later drop thereof) with respect to that skill. As expected, there were only a few reports of speaking experience in CS1 and CS2 – 10 in the latter. But nine additional students reported speaking experience in industry, English courses and/or other CS courses. Clearly, prior experience was a significant factor in the better than average initial confidence ratings of students for four of the communication skills.

VI. CONCLUSIONS AND FUTURE WORK

The central point of the work described here is to bridge the gap between CS1 courses, where communication activities are typically low-stakes, and software engineering and capstone courses, where communication is a major part of course content. We address this transition via a collection of assignment paradigms that can be used to seamlessly integrate communication skills with technical content.

The activities we describe are only starting points for integrating communication skills into the computer science curriculum, generally, and data structures and algorithms courses, specifically. We do acknowledge (and have experienced) that large class sizes, last-minute teaching assignments, and skeptical students (and colleagues) are challenges to this work. However, we strongly feel that inducing students to develop the communication skills required of professionals *along with the technical content of these courses* is well worth the time and effort.

In future semesters, we imagine several possible directions:

- *Tracking actual improvement in communication skills as the semester progresses and/or in follow-on courses*. There was an attempt to match the pre/post attitude survey in the course at one institution with another attitude survey in the capstone course, but there was no comparative *evaluation of actual communication skills*. Even the attitude survey did not allow for adequate comparison between students that were exposed to the above-mentioned assignments and students who were not; the questions were not coordinated and it could not be determined how many students (if any) took both surveys.
- *Evaluating the impact of communication assignments on technical skills*. There is a legitimate concern that these assignments require additional class time and/or reduce the portion of a student's grade dependent on technical competence. Anecdotal evidence suggests that increased emphasis on communication skills neither decreases nor increases technical competence. A more rigorous assessment of this observation would be useful.
- *Using professional-quality examples of work demonstrating the communication skills*. High-quality student work from previous semesters (anonymized) could serve this purpose. Speaking and teaming present difficult challenges. In case of the former, industry advisers have suggested use of examples provided by them; the process of developing a functioning team would probably have to be taught and supervised by a specialist, as is done at one of our institutions.

ACKNOWLEDGMENT

This work was funded by NSF CPATH-II Awards CCF-0939122 and CCF-0939081. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF).

References

[1] National Commission on Writing. 2004. *Writing: A Ticket to Work or a Ticket Out: A Survey of Business Leaders*. New York: College Board. http://www.collegeboard.com/prod_downloads/writingcom/writing-ticket-to-work.pdf.

[2] M. Carter, M. Vouk,G. Gannod, J. Burge, P. Anderson, M. Hoffman. "Communication Genres: Integrating Communication into the Software Engineering Curriculum," *ICSE*, 2011.

[3] G. McDonald and M. McDonald,. "Developing Oral Communication Skills of Computer Science Undergraduates," *SIGCSE*, 1993.

[4] J. Havill and L. Ludwig. "Technically Speaking: Fostering the Communication Skills of Computer Science and Mathematics Students," *SIGCSE*, 2007.

[5] L. Blume, R. Baecker,C. Cllins, and A. Donohue. "Communication Skills for Computer Scientists Course," *ITiCSE*, 2009.

[6] J. Hartman. "Writing to learn and communicate in data structures course," *SIGCSE*, 1989.

[7] D. Beard,D. Schwieger, and K. Surendran, K. "Incorporating Soft Skills into Accounting and MIS Curricula," *SIGMIS-CPR*, 2007.

[8] K. Falkner and N. Falkner. "Integrating communication skills into the computer science curriculum," *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, 2012.

[9] P. Gruba and H. Sùndergaard. "Constructivist Approach to Communication Skills Instruction in Computer Science," *Computer Science Education*, 2001.

[10] M. Hoffman, T. Dansdill, and D. Herscovici. "Bridging Writing To Learn and Writing in the Discipline in Computer Science Education," *SIGCSE*, 2006.

[11] C. Cilliers. "Student perception of academic writing skills activities in a traditional programming course," *Computers & Education*, 2012.

[12] R. Layton, M. Loughry, M. Ohland, and G. Ricco. "Design and Validation of a Web-Based System for Assigning Members to Teams Using Instructor-Specified Criteria," *Advances in Engineering Education*, 2010.

[13] G. Braught, J. MacCormick, and T. Wahls. "The Benefits of Pairing by Ability," *SIGCSE*, 2010.