Creating a Progression of Writing, Speaking, and Teaming Learning Outcomes in Undergraduate Computer Science/Software Engineering Curricula

Abstract

There are global concerns that communication (writing, speaking and teaming) skills of recent computer science/software engineering (CS/SE) graduates need improvement. This may be because instruction in communication is typically removed from the CS/SE curriculum, farmed out to courses focusing on general communication skills rather than those specific to CS/SE. However, it may be more effective to rely on the technical communication expertise of CS/SE faculty. We present preliminary results of a US National Science Foundation-funded project to design learning outcomes and teaching practices that will enable CS/SE faculty to integrate communication throughout their curricula. The approach is one of building a learning progression in communication based on genre theory and on the concept of exposing students to the appropriate responses to increasingly complex communication situations related to software development. The discussion begins with an analysis of a typical capstone course and then moves to analyses of lower-level courses.

Keywords: *Capstone Course, Genre Theory, Learning Progression, Undergraduate CS/SE Curricula.*

1. INTRODUCTION

The gap between the communication abilities of recent graduates and the expectations of managers in Computer Science and Software Engineering (CS/SE) and other engineering fields is well documented [(1), (7), (9), (14), (15), (16), (17), (18)]. One of the reasons this gap exists is that most instruction in engineering communication occurs outside engineering departments, typically in technical writing and oral communication courses. And even when communication is taught in engineering departments, it is typically done so by instructors trained in English or oral communication, not in engineering (15). Although these instructors may be experts in the general types of communication of the engineering workplace, they are usually not familiar with field-specific types of engineering communication [(15), (19)].

One solution to this problem is to integrate communication within the engineering curriculum. One of the most influential theories of learning to emerge over the last twenty-five years is based on research into apprenticeships (10). This research suggests that learning is most effective when it takes place in situations that are the same or similar to those in which learners will apply what they have learned, in other words, situations like apprenticeships. The key is that novices learn by doing what experts in the field do and novices learn through the guidance of experts, though without the high expectations and full responsibility of the experts. Called *Legitimate Peripheral Participation*, this theory suggests that people learn by engaging in the legitimate activities of the field in an environment in which those activities are normally done (10).

We suspect that most CS/SE faculty are more comfortable with the apprenticeship model of learning in regard to technical skills than communication skills. This may be because they do not consider themselves as experts in communication. However, because CS/SE faculty are trained in the discipline and often have industry experience, they have a far greater expertise in the types of communication that professionals in the field do than instructors in English and oral communication [(6), (19)]. In terms of the apprenticeship model, technical skills and communication skills are on an equal footing and CS/SE faculty are expert in both. If CS/SE students are to become capable as professionals in their field, they need to receive instruction and experience in communication in the field. So communication should be integrated into technical education to the same extent as the two are integrated in professional environments in CS/SE.

One major hurdle in achieving this goal is *how* to integrate communication into the CS/SE curriculum. That is what this paper is about. We offer strategies for identifying the types, or genres, of communication that are specific to the field and integrating these types of communication in a way that promotes growth in abilities throughout a curriculum. We also provide an example of the application of these strategies to a CS/SE curriculum, more specifically, as related to the expression of code design. We start by analyzing the capstone course and then we analyze two sequential introductory courses, which we call CS1 and CS2.

2. GENRE IN COMPUTER SCIENCE AND SOFTWARE ENGINEERING

To be able to provide coherent instruction and communication experiences for students in CS/SE, we need to identify the specific genres of communication that define the field. In a classic definition by Miller (11), genres are "typified rhetorical actions based on recurrent situations." In other words, certain categories of communication develop in response to situations that recur with some frequency. This concept may sound familiar to computer scientists because it is similar to design patterns associated with object-oriented software (6). Both are reusable forms or templates that can be applied to commonly recurring situations.

CS/SE is characterized by many recurring situations that have led to a set of typical responses, or forms of communication, to those situations. In one example of such a situation, a person needs to communicate what a client requires for solving a problem to a software engineer who is not familiar with the client or the client's problem. The communicator must respond by providing the information necessary for the software engineer to successfully solve the client's problem. The form of communication that has developed to address this situation is software requirement specifications (SRS), which describes in user terms what a program should do. Over time, the SRS has come to be defined by a set of conventions that the communicator can call upon to achieve the particular purpose for the particular audience in this recurring situation (e.g., SRS can be provided in terms of functional or non-functional requirements, use cases or user stories, etc.). As another example, the expression of a design may be understood as a typical response to the recurring situation in which functions necessary for the software to appropriately perform need to be explained in a way that provides a blueprint for implementing code.

Though it is not possible to account for every recurring situation and communication response in the field, there are some that are classic (see Table 1). As the list of genres in Table 1 suggests, communication that is specific to CS/SE plays a critical role in the apprenticeship of students in the field. It introduces them not only to the individual situation and responses that define the field but also to the broader software development life cycle consisting of these genres. The responses are a part of the legitimate participation of people in CS/SE, integrated with the technical production they are asked to do. In contrast to Orr's (12) list that includes more general professional genres—e.g., progress reports, trip reports, and technical reports—we have focused on those that are specific to CS/SE. We also make a distinction between genre and medium. Whereas genre is the recurring CS/SE situations and communication responses to those situations, medium is the *means* by which we communicate, such as email, phone calls, PowerPoint presentations, and whiteboards.

Table 1. Common Genres in Software Engineering

Definition of a SE problem	Test plan
SRS	Testing
Design expression	Bug report
Code and comments in the code	Installation and Maintenance guide
Developer guide	User guide

3. Growth in Communication Throughout the CS/SE Curriculum

The concept of legitimate peripheral participation proposed by Lave and Wenger provides a model for growth in student learning that could be used to shape a curriculum that integrates communication (10). Students enter our programs as novices in CS/SE, on the periphery of the field. The goal is to move them toward full membership as professionals in the field. In most CS/SE curricula, the capstone course is a bridge between the classroom and the workplace (13). In many of these courses, students are placed in situations that mimic those they will encounter as professionals. The curriculum, then, should be designed to develop students as professionals before entering the capstone, with the technical and communication skills they need to succeed.

There are two principles that guide this development of students as capable communicators in CS/SE (6). One is that students should participate in the legitimate communication practices of the field, from first through senior years. Implicit in this principle is that there is no inherent hierarchy of genres that would lead to a stepped progression from one to the other. Development in communication is not a matter of first mastering one genre and then another in stepwise fashion through the list of CS/SE genres. Rather, development depends on students having ample opportunities to communicate in a broad range of disciplinary genres throughout their programs.

The second principle is that progressive development as communicators in a field is a function of the increasing sophistication of student responses to the increasingly complex situations of assignments. The idea here is that the students' communication becomes more sophisticated as the situations they are responding to become more complex. For example, they may be asked to write a simple, perhaps even one-sentence, SRS in CS1 and develop progressively in writing more and more sophisticated SRSs, culminating in the capstone course. But the question is what does that growth in sophistication look like given the need to address multiple genres across an entire curriculum? How do CS/SE faculty create progressively complex situations throughout a curriculum? Here we offer one way of conceiving complexity that can be used to promote growth in sophistication of students' communication.

The first principle suggests that we should incorporate all genres at all levels of the undergraduate curriculum. The second principle suggests that the situations and responses increase in sophistication as the student moves through the curriculum. We have integrated these principles into five dimensions of situational complexity that can guide development of a learning progression. We apply these dimensions of situational complexity in sections 4 and 5.

The five dimensions of situation complexity are that:

1. More complex situations require longer responses.
2. More complex situations require more complicated responses.
3. More complex situations require responses in multiple genres.
4. More complex situations require a greater degree of independence on the part of students.
5. More complex situations require more students to contribute to the completion of a project.

4. CREATING A LEARNING PROGRESSION FOR A CS/SE CURRICULUM

As we have suggested, Lave and Wenger's (10) model of apprenticeship offers a way to conceive the development of CS/SE students as they move from novice toward expertise in the field. Students learn by doing what experts in the field do, which includes communication. In this section, we describe a method for shaping a curriculum that integrates communication in a way that also advances the technical abilities of students. The method consists of two stages, first identifying program learning outcomes for technical and communication abilities and second shaping a curriculum for developing students as effective communicators in the field.

Identifying program learning outcomes is important because planning on the curricular level requires that faculty understand what students should be able to do by the time they graduate. The idea is that a student who graduates with a CS/SE degree may be distinguished in his or her preparation from a person who is simply good at programming. A key marker is that the graduate brings particular *ways of thinking* about CS/SE that the other person typically does not possess. Faculty at different CS/SE departments may have different descriptions of what all of their students should be able to do by the time they graduate. As an example, our faculty produced the list in Table 2 (the learning outcomes related to the science of computing are not included here so as to focus on software development).

Table 2: Selected Program Learning Outcomes Defining the Ways of Thinking of Software Development

To demonstrate that graduates can develop software, they should be able to:

1. Recognize/define a problem related to a specific scenario that can be solved with a software application. Describe how the end-users or internal actors within a system intend to use the application to be developed. Gather and analyze information that allows for requirements that will solve the problem to be created, validated, verified and if necessary, revised.
2. Create/express a design for an underlying abstract model of computation that accommodates defined system requirements—including considerations of privacy, security, and efficiency—so that a developer can implement the application. Review the design to ensure it can accomplish the requirements and, where it does not, redesign until it meets the requirements.
3. Implement software conforming to a specified design so that it is usable, testable and modifiable by others. Review the implementation to ensure it meets the system requirements and conforms to design and, where it does not, correct the implementation until it meets the requirements and design.
4. Plan/execute appropriate tests in order to identify ways in which the software does not meet the requirements and, where it does not, to redesign, implement and retest until it meets the requirements.

How do students develop these ways of thinking? Communication plays an essential role because the different genres of CS/SE shape particular ways of thinking, both individual genres and the collective genres that define the software development cycle. Thus, the next step after describing the technical ways of thinking in CS/SE is to specify the communication abilities that go hand in hand with those ways of thinking. An example created by our faculty is in Table 3. Students both learn and demonstrate that they have learned the technical ways of thinking by engaging in these forms of communication. For instance, the ability of students to "Recognize and define a problem" in the technical outcomes is learned and demonstrated by students through the communication outcome in which they are expected to "Present in writing or orally a critical assessment of a problem situation defined by a need for software to be developed for solving the problem…."

Table 3: Selected Communication Outcomes of Software Development for Students to Learn and to Demonstrate that They Have Learned the Ways of Thinking in Table 2.

To demonstrate that graduates have achieved the general program learning outcomes, they should be able to:

1. Present in writing or orally a critical assessment of a problem situation defined by a need for software to be developed for solving the problem: (a) collect information from sponsors, end-users, and on-site observations, (b) analyze that information (c) use the analysis to define the problem in terms of the stakeholders' needs and goals for addressing those needs.
2. Write requirements representing the stakeholders' needs and goals in such a way that the requirements can be applied in a design by others.
3. Read requirements for various purposes, such as to inspect and correct them, to validate them as meeting the user's needs, to revise them so that they better meet user's needs, to implement them in a design, and to identify what students don't know and what they need to know to create code.
4. Write a design that accommodates the defined system requirements—including considerations of privacy, security, and efficiency—so that a developer can implement the application.
5. Read a design for various purposes, such as to ensure it can accomplish the requirements and, where it does not, redesign until it meets the requirements and to translate it into code.

Table 3 (Continued): Selected Communication Outcomes of Software Development for Students to Learn and Demonstrate that They Have Learned the Ways of Thinking in Table 2.

To demonstrate that graduates have achieved the general program learning outcomes, they should be able to:
6. Write a program to conform to a specified design so that it is usable, testable, and modifiable by others.
7. Write a narrative description of code, including a list of file names or directories included.
8. Read code and comments for various purposes, to find and correct errors in syntax and semantics, to determine what a program is supposed to do, to revise a program so that it accomplishes what it is supposed to do, to modify a program for different purposes, to ensure that a program conforms to system requirements and conforms to design, to provide productive feedback to those who created it, to continue a program begun by someone else, and to apply it to new uses.
9. Write a developer guide that is appropriate to the audience.
10. Write a user guide that is appropriate to the audience.
11. Present in writing or orally a test plan and results of testing that identifies ways in which the software does not meet the requirements.
12. Work effectively in teams: (a) develop ground rules to guide the team's approach to work; (b) define roles so that expectations of team members are clear and followed, (c) create agendas and minutes for team meetings; (d) interact with other team members in ways that assure the productive contributions of all team members; (e) create specific action items for each member and then hold him or her accountable; (f) identify, create, and manage the tools that enable teams to work effectively; (g) resolve conflicts among team members.

The next stage is the analysis and revision of a curriculum by which students develop the expected ways of thinking and communicating. This process may be done in four steps.

1. Align the capstone course with the program outcomes. As we have noted, the typical role of the capstone is to provide a bridge for students between the university and the workplace. It is where students engage in activities that encourage them to synthesize and apply what they have learned in a context that mimics the workplace. Thus, ideally, a capstone would reflect the ways of thinking and communicating in the program outcomes. So addressing the curriculum would begin with a review of the capstone in light of the outcomes and, where it is found wanting, to revise it.
2. Analyze the capstone for complexity of its communication assignments. If the capstone represents the ultimate student experience in preparation for the workplace, then it also serves as a target for the preparation of students earlier in the curriculum. A curriculum should be structured so that it enables students to develop the technical and communication abilities necessary to take full advantage of the opportunities in the capstone by extending the ability they have gained in previous courses. We have defined development of communication abilities as a growth in the sophistication that is guided by the increasing complexity of communication assignments. Therefore, as a target for the development of students in the curriculum, the capstone's assignments need to be analyzed for their complexity in the five dimensions we have presented.
3. Assess the present curriculum for preparation of students for the capstone. This assessment is necessary for revising the curriculum. The question to be answered is where in the curriculum are students engaging and receiving instruction and guidance in the genres they must use in the capstone? That information could be derived from a survey of faculty and/or students or an analysis of syllabi. The first principle we discussed above suggests that students should be given opportunities to participate in the genres of the field in courses at

all levels. The goal of this assessment, then, is to identify gaps in students' preparation, what genres are missing or are being under taught. For practical purposes, it may be simpler at first to focus only on the required courses in the curriculum.

4. Revise the curriculum so that it better prepares students for the capstone. Any gaps for genres in the curriculum exposed by the assessment are opportunities for improvement. Where there are genres missing or used in only one course, where else should they be used? The goal is not just to determine where genres are to be included but also the growth in sophistication of students' work in those genres. It is important that the appropriate levels of complexity of assignments be considered. For example, students may experience a genre for the first time in an introductory course by reading it rather than writing it. In that instance, instruction would be designed to make students aware of the name of the genre, its role in the development of software, and perhaps its features. In this way, many genres may be introduced to students in their first required course to initiate their development in the field.

5. Creating a Learning Progression: An Example

In this section, we outline a sample learning progression using the complexity theory defined in Section 3 and the Design Expression genre.

For step 1, we determined the alignment of our capstone course to the program learning outcomes (see Table 2). Our capstone course focuses primarily on software development, and we found that it encompasses all the ways of thinking and communicating associated with that area (outcomes 1-4 for the former and 1-12 for the latter). Most students do a standard software development project in the course.

For step 2, we created a table by which we analyzed the complexity of the communication genres of the capstone course [see (5)]. Students begin with a problem paragraph and over 16 weeks create a solution to the presented problem. Working as teams inside and outside of class, the students iteratively explore requirements, express and evaluate designs, and implement and test a coded application that solves the original problem.

For step 3, we used an informal survey given to capstone students on the first day of the course [see (5)]. The primary purpose of the survey was to gauge the level of students' preparation in communication in the field in order to better meet their needs. We found that their experience with the genres of the field took place almost exclusively in Software Engineering.

Our Software Engineering course requires students to work in teams to develop additional system requirements and create related designs as they build onto a larger code base. They are also expected to test their section of the code as well as the entire code base. In semesters past, this is first time that students had been exposed to these various CS/SE genres.

As a result of the conclusion of step 3, for step 4, we focused primarily on our CS1 and CS2 courses because there was little if any work with the targeted genres in those courses. The challenge is to engage students in communication in the field at the appropriate level. In early years, we recognized that one of the simplest ways to create a connection from course to course was to use the same language; instructors of CS1, CS2, Software Engineering and the Capstone focused on providing consistent genre definitions across the curriculum.

As a part of a US National Science Foundation (NSF)-funded study, over the past two semesters, we have been in the process of modifying CS1 and CS2 courses so that a learning progression of communication skills is realized. Table 4 shows an example of the evolution throughout a curriculum of teaching students about Design Expression. In CS1, for example, students are expected to create a program from a design already prepared for them (note in rare cases, the students may tweak the design slightly through the addition of “helper methods” to finish the assignment). After students are introduced to model-view-controller (MVC), the instructors provide an explanation of earlier design decisions in terms of MVC.

As students progress through the curriculum to CS2, all projects have a design phase where students are expected to create their own design and discuss rationale for their decisions made during that process. This presentation includes the creation of diagrams that focus on low-level design and related written narratives. In some cases, this is done in two-person teams. After the initial design stage, complexity is controlled by requiring code development to conform to instructor-provided designs. Upon completion of the project, students reflect on the similarities and differences between their design and the instructor’s design. Some projects may require a design inspection between teams in class.

The opportunities given to students to express code design in a well-developed CS/SE capstone course are also presented in Table 4. Note that students are expected to work on a team throughout the semester, creating and changing their design based on evolving system requirements. The expression of this design takes different forms – they are expected to create diagrams and related written narratives, as well as present/evaluate their design (in both oral/written forms) with mentors and peers. The team’s design serves as an outline for them to assign individual implementation assignments to each team member.

The sophistication of the required communication in the capstone experience is imperative to the development of budding computer scientists. By creating an appropriate learning progression in the lower-level courses, the opportunities presented in the capstone experience become richer—it will not be the first time students are exposed to design evaluation, for example. As a result of being exposed to reflecting upon their own design in CS2, they have a point of reference upon which to help adapt their team’s design in response to evolving system requirements in their capstone experience.

Table 4. Sample Learning Progression/Complexity Analysis Using Design Expression Genre.

Responses are…	**CS1**	**CS2**	**Capstone**
Increasingly longer based on extent of engagement	Students are provided the design & must follow the design.	Students create their own design of a system & must justify the design decisions they made. When implementing code, the students are provided a design.	Expression of design accommodates SRS and is usually created iteratively; often begins as diagram; evolves to multiple diagrams with accompanying descriptive narrative.
More complicated	Students are provided the design & must follow the design.	Students create their own design of a system & must justify the design decisions they made. May include design inspection.	Changes throughout course of development to accommodate need for effectiveness & efficiency.
Related to multiple genres	Students are provided the design & must follow the design.	Students create design based on provided requirements.	Design accommodates requirements & serves as roadmap for code development.
Created independently	Students are provided the design & must follow the design.	Students submit their own design, but implement teaching staff design. Students provide written reflection on two designs.	Students are often required to create designs independently of mentors; must defend design decisions when presented to mentors/peers.
Created by teams in various venues	Students work independently. May create other design elements.	Students may work independently or in pairs/teams. Students work outside of class.	Design processed & agreed upon by entire team; provides vision for all & launching point for individual implementation.

6. Conclusion

We have described and presented an example of a process for enhancing CS/SE students learning by integrating communication in the curriculum. This process is based on the idea that communicating in the genres of a discipline promotes learning of both technical and communication skills in the discipline [(2), (3), (4)]. The apprenticeship model we have applied here suggests that to learn effectively, students should participate as fully as appropriate in the activities of experts in the field under the guidance of those experts (10). In the academy, it is the professors in each discipline who function as the experts, providing students instruction and experiences designed to develop the technical and communication abilities of the disciplines. In CS/SE, this means creating learning situations similar to those students will encounter as professionals in the field, situations in which technical and communication deliverables are integrated.

This integration could elevate the profile of communication in CS/SE. It becomes a field in which excellence in communication is valued along with excellence in technical skills.

Changing the perception of CS/SE to a discipline that values communication along with technical skills, one that stresses social interaction rather than individual work, could have the effect of attracting students who are more comfortable with that approach, such as women and underrepresented minorities.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CCF-0939122.

References

(1) Bates, F. E. and Connor, D. A. Industry Survey for the University of Alabama at Birmingham (UAB): 2005 Electrical Engineering Curriculum Survey. *Proceedings of the 24th Annual Frontiers in Education Conference.* (2-6 November, 1994), pp. 242-255.

(2) Carter, M. Ways of Knowing, Doing, and Writing in the Disciplines. *College Composition and Communication.* Vol. 58, No. 3, (February, 2007), pp. 385-418.

(3) Carter, M., Ferzli, M. and Wiebe, E. Teaching Genre to English First-Language Adults: A Study of the Laboratory Report. *Research in the Teaching of English.* Vol. 38, No. 4, (May, 2004), pp. 395-419.

(4) Carter, M., Ferzli, M. and Wiebe, E. N. Writing to Learn by Learning to Write in the Disciplines. *Journal of Business and Technical Communication*. Vol. 21, No. 3, (July 2007), pp. 278-302.

(5) Carter, M., Fornaro, R., Heckman S. and Heil M. Developing a Learning Progression that Integrates Communication in an Undergraduate Computer Science/Software Engineering Curriculum, *North Carolina State University Technical Report*, TR-2012-XX, May 25, 2012. http://www.csc.ncsu.edu/research/tech/reports.php#2012.

(6) Carter, M., Gannod, G., Burge, J., Anderson, P., Vouk, M. and Hoffman, M. Communication Genres: Integrating Communication into the Software Engineering Curriculum. *Proceedings of the 24th IEEE-CS Conference on Software Engineering Education and Training*, (22-24 May, 2011), pp. 21-30.

(7) Ford, J. D. and Riley, L. Integrating Communication and Engineering Education: A Look at Curricula, Courses, and Support Systems. *Journal of Engineering Education,* Vol. 92, No. 4, (2003), pp. 325-328.

(8) Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley. 1994.

(9) Henderson, K. Educating Electrical and Electronic Engineers. *Engineering Science and Education Journal*. Vol. 6, Issue 3, (1997), pp. 95-98.

(10) Lave J. and Wenger, E. *Situated Learning: Legitimate Peripheral Participation*. Cambridge: Cambridge University Press. 1991.

(11) Miller, C. R. Genre as Social Action. *Quarterly Journal of Speech.* Vol. 70, Issue 2, (1984), pp. 151-167.

(12) Orr, T. Genre in the Field of Computer Science and Computer Engineering. *IEEE Transactions on Professional Communication.* Vol. 42, No. 1, (1999), pp. 32-37.

(13) Paretti, M. C. Teaching Communication in Capstone Design: The Role of the Instructor in Situated Learning. *Journal of Electrical Engineering.* Vol. 97, No. 4, (2008), pp. 491-503.

(14) Pinelli, T. E., Barclay, R. O., Keen, M. L., Kennedy, J. M. and Hecht, L. F. From Student to Entry-Level Professional: Examining the Role of Language and Written Communications in the Reacculturation of Aerospace Engineering Students. *Technical Communication.* Vol. 42, No. 3, (1995), pp. 492-503.

(15) Reave, L. Technical Communication Instruction in Engineering Schools: A Survey of Top-Ranked U.S. and Canadian Programs. *Journal of Business and Technical Communication.* Vol. 18, No. 4, (2004), pp. 452-490.

(16) Riley, L. A., Furth, P. M. and Zellmer, J. T. Assessing Our Engineering Alumni: Determinants of Success in the Workplace. *Proceedings of the 2000 ASEE/Gulf-Southwest Sectional Annual Conference*. (2000), Section 73A1.

(17) Sageev, P. and Romanowski, C. J. A Message from Recent Engineering Graduates in the Marketplace: Results of a Survey on Technical Communication Skills. *Journal of Engineering Education.* Vol. 90, No. 4, (2001), pp. 685-697.

(18) Vest, D., Long, M. and Anderson T. Electrical Engineers' Perceptions of Communication Training and Their Recommendations for Curricular Change: Results of a National Survey. *IEEE Transactions on Professional Communication.* Vol. 39, Issue 1, (March, 1996), pp. 38-42.

(19) Wolfe, J. How Technical Communication Textbooks Fail Engineering Students. *Technical Communication Quarterly.* Vol. 18, No. 4, (September, 2009), pp. 351-375.