CS/SE Instructors Can Improve Student Writing without Reducing Class Time Devoted to Technical Content: Experimental Results

Paul V. Anderson
Elon University
Elon, NC, USA
panderson4@elon.edu

Sarah Heckman, Mladen Vouk, David Wright, Michael Carter
North Carolina State University
Raleigh, NC, USA
{sarah_heckman, vouk, david_wright, mpc}@ncsu.edu

Janet E. Burge
Wesleyan University
Middletown, CT, USA
jburge@wesleyan.edu

Gerald C. Gannod
Miami University
Oxford, OH, USA
gannodg@miamioh.edu

***Abstract*— The Computer Science and Software Engineering (CS/SE) profession reports that new college graduates lack the communication skills needed for personal and organizational success. Many CS/SE faculty may omit communication instruction from their courses because they do not want to reduce technical content. We experimented in a software-engineering-intensive second-semester programming course with strategies for improving students' writing of black box test plans that included no instruction on writing the plans beyond the standard lecture on testing. The treatment version of the course used 1) a modified assignment that focused on the plan's readers; 2) a model plan students could consult online; and 3) a modified grading rubric that identified the readers' needs. Three external raters found that students in the treatment sections outperformed students in the control sections on writing for five of nine criteria on rubrics for evaluating the plans and on the raters' holistic impression of the students' technical and communication abilities from the perspectives of a manager and a tester.**

***Index Terms*—communication across the curriculum, software engineering education, black box test plans.**

I. Introduction

Computer Science and Software Engineering (CS/SE) are professions in which technical and communication abilities are essential to personal and organizational success [2, 12, 23]. In almost every professional setting in these fields, the technical work that one person performs must be communicated to another person who needs that result in order to perform his or her own job [2, 12]. Poor communication creates costly mistakes. Time is wasted. Products function less well than they should. Clients, customers, and others who depend on the potential power of CS/SE solutions are frustrated.

A common complaint among employers is that students graduating from CS/SE programs are generally strong in their technical knowledge but notoriously weak in their ability to communicate effectively in the CS/SE workplace [2, 9, 23]. The generally low state of a college graduate's workplace communication abilities is attributed to many causes. Faculty say that they do not have the training to provide writing instruction in their classes [20, 23]. They add that if they take class time to explain and demonstrate professional communication, they will be placing their students at a disadvantage by spending less time on essential technical content [12].

In this paper, we describe an experiment in which we tested two versions of an approach to developing CS/SE students' communication abilities that requires no special knowledge of writing pedagogy by the teaching staff and takes no class time away from the technical instruction. This experiment continues a line of research that integrates communication tasks into CS/SE courses by embedding students' technical work in scenarios that resemble situations in which the same work is often performed in the workplace: when a person completes a technical task (e.g., a bug review), he or she must communicate the results to persons who need the results in order to perform their own work (e.g., correcting the bugs) [5, 6, 8]. We have named this kind of instruction Reader-oriented Integrated Technical and Communication Instruction (RITCI).

We conducted our experiment in a software-engineering-intensive second-semester programming course for undergraduate computer science majors and minors at NC State University. In the control and experimental versions of the course, students prepared black box test plans without receiving any instruction on writing the plans beyond a reading and lecture on software testing. Our experimental results suggest that it is possible to improve students' abilities to create a communication artifact that meets the needs of its readers without taking class time for communication instruction by focusing on audience needs in the assignment and rubric, and by providing a model document. Our results also suggest these same strategies can increase students' abilities to understand and apply a course's technical content.

The rest of the paper is organized as follows: Section 2 discusses related work; Section 3 presents our research hypotheses; Section 4 describes our research method; Section 5 describes the results; Section 6 discusses the results and their implications for CS/SE communication instruction; Section 7 identifies the threats to validity; and Section 8 describes the conclusions drawn from the study.

II. Related Work

Researchers have speculated that integrating communication and technical content instruction in courses could enhance students' abilities to learn both. One line of reasoning has emphasized the value of teaching students to communicate the results of their technical work within the genres of the field [9, 11]. Learning the genres of a field is equivalent to learning how specialists in the field think and do their work [7]. Another line of reasoning has emphasized the value of creating scenarios in which students need to convey the results of their technical work to the people who would need to use the results to complete their own work [5, 6, 17]. The two approaches overlap and reinforce each other. Genres are recognized as a conventional way of addressing recurring communication needs in ways that guide writers and meet readers' needs [9, 11]. Scenarios involve the need to communicate specific information from particular technical work to the specific persons who need to use it [17]. Instruction using both genres and scenarios is part of legitimate peripheral participation or situated learning [21], a pedagogical technique where students engage with materials in similar ways that they might in industry [2, 5].

The joining of scenarios and genres can be particularly powerful in fields such as CS and SE. Some fields, such as the natural sciences, have a few iconic genres, such as the scientific research report and the literature review. In contrast, CS and SE have many more genres [9, 11]. The difference has partly to do with the ways that groups of people work together to achieve a particular outcome in CS and SE. In the natural sciences, for instance, there may be many contributors to a scientific research project, but all work together to create a single report, which in some cases may list dozens of authors. In contrast, in CS and SE, many actors perform their own specialized work and pass the results along to another person who must use the first person's results to perform his or her own work. In a sense, CS and SE work is like a relay race, with each person running part of the race, then handing the baton to the next person. At each handoff, there is a conventional way of communicating—a genre—that serves its particular readers only if it meets their needs in the special circumstances of the project on which the writer and reader are working.

While the integration of performing technical work and communicating about it to others who need to use the results is a ubiquitous feature of the workplace, technical and reader-oriented communication tasks are usually combined in only a few places in CS/SE curricula, primarily the software engineering and capstone courses [6]. Reave [23] provides a survey of communication in engineering disciplines that describes the high frequency of stand-alone communication courses and low incidence of integration of communication within engineering courses. However, Burge, et al., [6] found that only 37% of CS/SE undergraduate programs, out of 104 analyzed, required a stand-alone communication course and that less than 15% indicated a strong emphasis on writing skills in technical courses other than software engineering (54%) and the capstone (73%). Several CS/SE specific frameworks for incorporating communication interventions into courses and curricula have been proposed [6, 8, 12, 17, 20]. In these frameworks, assignment development utilizes situated learning, where technical tasks are framed within audience-driven workplace scenarios [2, 17, 20]. The reader requires the technical communication to complete a task. The studies describing these frameworks focus on assignment design, without saying much, if anything, about the associated instruction. The study reported here, which focuses on instruction, suggests that better student writing can be achieved without devoting classroom time to communication instruction if the assignment includes a reader-focused workplace scenario, uses a reader-sensitive rubric, and is accompanied by a model document that illustrates ways to meet the reader's needs.

III. Research Hypotheses

In this article, we describe an experiment using the RITCI approach, for which we tested the following hypotheses.

- Using RITCI in a CS/SE course without taking class time away from technical instruction will have a positive impact on the quality of students' communication abilities.
- Using RITCI in a CS/SE course without taking class time away from technical instruction will have a positive impact on the quality of students' technical abilities.

IV. Method

Our experimental method involves comparison of black box test plans, as a project management genre [11], written by students in two semesters of a second-semester programming and software engineering course, CSC216: Programming Concepts - Java. CSC216 integrates software engineering processes and practices. The course learning outcomes cover introductory software engineering (design, unit and system/acceptance test, code coverage, static analysis, and the software development lifecycle); advanced object-oriented programming skills (composition, inheritance, interfaces, abstract classes, and polymorphism); linear-data structures; recursion; and finite-state machines. Such a course is common in both CS and SE programs. CSC216 serves as an introduction to fundamental software engineering practices and skills that students build on in later software engineering courses. Students complete three multi-part projects as part of their coursework in CSC216.

A. Student Artifacts Tested

The initial part of each of the three projects includes the creation of a black box test plan, which students are assigned to write before they begin coding their program. We chose to focus on the genre of system or acceptance tests modeled in a black box test plan document because it is an essential element in the process of program or system design in many professional settings and because it has several distinct parts, each with a widely accepted purpose and type of content [4, 11]. The black box test document is often presented as a printed or online table or form in which the writers are prompted to enter certain kinds of information to define each test case for a system.

Students in both the control and treatment sections were asked to describe their plans in the same form or template, which instantiated the generic conventions of black box tests. In addition to lines for recording the "author(s)" names and date, the form had three sections: an Introduction section for providing an overview of the black box test plan, a four-column table for entering information about each test (Fig. 1), and a table for recording the revision history of the document.

The four-columns of the table provide the structure for articulating a test case. In some organizations, a form like this one is completed by two people. The first writer is the system designer or programmer, who fills in the introduction and three left-hand columns. Especially important are the middle columns, which provide instructions for testing the finished product. Typically based on the requirements specification, entries in these columns describe the test cases and provide the expected or desired result for each. The first writer completes his or her entries before design and coding work begins.

Typically, the first writer then passes the black box test plan to a testing specialist, who uses the instructions to execute the test cases to determine whether the expected outcomes are produced. Usually new employees also submit their test plans to a manager who reviews the test cases against the requirements to assure that all critical functions of the program or system will be tested.

While the form seems like a straightforward document to fill in, the first writer can write it very effectively or ineffectively from the tester's as well as the manager's perspective. If a test plan is written poorly, the manager and tester may both have to ask for corrections or additional information, or they may assume that something important is being tested when it is not. Even a good test plan can be poorly written so that the manager misunderstands it or so that the tester cannot correctly conduct the tests – both highly undesirable outcomes. The outcomes of the tests are important to the organization, which depends on them to measure the current quality of the system. The tests in the plan can also be well or poorly chosen, according to whether the range of critical actions by users are or are not included. Managers and others use the test results to determine if the product is completed; if it is not, they will use the results to determine how much more work will be required and to identify the specific execution(s) that lead to failure. Therefore, for both the treatment and control versions of the course, we embedded the creation of black box test plans into scenarios in which the plans were to be read by a manager and a tester.

Test ID	Description	Expected Results	Actual Results

Fig. 1. Table from the form for describing individual black box tests

The sample student plans that were scored by the raters were deidentified in ways that prevented the raters from knowing the students' names, the course sections they were in, and the year in which they took the course (control year or treatment year). Forty-five black box test plan artifacts were considered for the experiment; 31 from Fall 2011 and 14 from Fall 2012 [24]. Evaluated artifacts were selected from all of the submitted black box test plans by Ohio's Evaluation and Assessment Center for Mathematics and Science Education [24]. For both the control and treatment groups, the raters considered only black box test plans from the first two of the three projects in each semester.

B. Research Participants

The students who participated in this study were volunteers from the on-campus sections of CSC 216, Programming Concepts - Java, taught in Fall 2011 and Fall 2012 (NC State University IRB #1749). Students registered for the courses followed standard registration procedures. No special grouping was used. The control group took the course the year before the treatment group. Most students were taking the course for the first time, but several students were repeating the course. One student was common between the control and treatment groups, but we do not know if the student's artifacts were evalauted in the study. Two instructors each taught one section of CSC216 in Fall 2011. The two instructors collaborated on creation of the projects that were administered to both on-campus sections. One of the instructors from Fall 2011 taught both on-campus sections of CSC216 in Fall 2012.

Control Group. The control group consisted of 115 students in two on-campus sections of CSC 216 taught in Fall 2011. The projects set up the scenario in which students were to play the role of employees of a software company that was designing new programs. During the semester, the students worked sequentially on three small programs, each of which they completed in five to ten Java classes. In each instance, the students were told that their manager asked them to write a black box test plan for the program they were creating. For the first assignment, students created at least one test case. For the second and third assignments, students created at least three test cases.

Treatment Group. The treatment group consisted of 134 students in two sections of CSC 216 taught in Fall 2012. The Fall 2012 offering of the course also involved creation of three programs. The programs that the students in the treatment group were to write black box text plans for were different and, for reasons unrelated to this study, more complex than the ones for the control group. As in the control sections, the three projects used the scenario in which students were to play the role of employees of a software company that was implementing new programs. Students worked sequentially on three small programs, each requiring five to ten Java classes. In each instance, they were told that their manager asked them to write a black box test plan for the program they were creating.

For the first assignment, they were to create at least one test case. For the second and third assignments, they were told to create at least three test cases.

C. Experimental Interventions

For the treatment group, the instructor modified two documents associated with the test plan assignments and created one other. No class time beyond that used in this instructor's control section was devoted to communication instruction aside from telling the students that the new document was available on the course website. In presenting the three interventions, we first describe what the control and treatment versions had in common and then identify the changes for the treatment group.

Modified Printed Assignments. The assignments for the control and treatment groups were very similar. Both used identical words to explain why the manager wanted the test plans.

> These tests will be used to determine if the finished product is ready to ship to the customer. Your manager wants you to write the tests before you begin development so that testing may be ongoing during development. Writing the tests will also help clarify the inputs and output to the system.

The assignments for both groups also provided the same, brief guidance for telling the tester how to set up the application for testing.

> [Y]ou must provide instructions for how a tester would set up the application for testing (*what class from your design contains the main that you would use to start your program*). [emphasis added]

The assignments for the treatment group differed in small but significant ways when describing the specific tests to be run. For both groups, the assignment stated that the tests must be repeatable and specific; the inputs and expected results values should be concrete allowing for replication of the tests. However, the treatment assignment included a statement that each test "must demonstrate that the program satisfied a scenario from the requirements." The emphasis on scenarios requires students to demonstrate a deeper technical understanding of the different ways that a customer might use a software product and to demonstrate mastery of technical testing topics like using equivalence classes and boundary values for identifying test inputs.

The framing of the students' assignment as described above satisfies all but one of the elements of RITCI. The students' technical task (i.e., creating one or more black box test(s)) is embedded in a scenario that places them in a workplace setting and asks them to write the kind of communication that professionals in that setting are sometimes asked to prepare. The assignment for the control group is missing the reader who must use the student's communication to perform his or her job. In fact, an additional note in the control assignment suggests that the students themselves will be the real user of their communication: "When you submit working code, you must also document the actual results of running your tests on your finished program." To emphasize the importance of writing for others, this direction might have explained that at the end of the project, the students will switch roles from that of the test planner to that of the tester in order to see how well their code and their test plan worked. Without such an emphasis, the control sections constitute a non-RITCI version of the course.

For the treatment group, the revised assignment placed greater emphasis on the tester as a reader who must use the test plan, making the treatment sections into a RITCI version of the course. It supplemented guidance about the technical details (inputs, expected results, values) with details about the tester and what the tester needs in order to conduct the test:

1. The importance of understandability by a specific kind of reader was added: "Write the instructions so anyone using Eclipse could run your tests."
2. Additional information the reader needs to conduct the test was identified: "What the input files contain. The command line arguments, if any."
3. The importance to the reader of complete information was implied: "All inputs required to complete the test must be specified."

The addition of this guidance about the reader and the reader's needs shifts the directions for conducting the tests from merely providing technical details to emphasizing the way to communicate the details so that another person can use them to perform his or her own work, which, in this case, is to execute the tests. The technical and communicative tasks are integrated, as would be the case in the workplace [2].

Model Test Plan. Perhaps the most significant intervention for the treatment group was the creation of a model test plan that illustrated how to write in a manner beneficial to the reader (tester). The control group had no such model. The features of the model for the treatment group that were most likely to have been influential were located in the introduction and the table for describing individual tests.

Introduction. The only guidance for writing the introduction that the control group received was the following pair of sentences, printed under the heading for the section: "Provide a brief overview of your black box test plan. Describe how *you* would start *your* application." [emphasis added].

For the control group, the terms "you" and "your" in the second sentence suggest to students the default situation in most CS classes (including this one): they are writing for themselves and for the instructor. Because this guidance appears in the very form where students write their introductions, the "you" and "your" could erase from students' memories the recollection of the single reference to the tester in the assignment. For the control group, there is no other guidance to remind them that they are writing for the tester.

In contrast, the model test plan given to the treatment group presented the process for starting the application as step-by-step directions addressed to another person.

> If using Eclipse, first create a Run Configuration for the project by right clicking on SomewhatSimplifiedSolitaireUI and selecting Run > Run Configuration. Select the new configuration button in the upper left. Browse for the SomewhatSimplifiedSolitaire project. The main class is edu.ncsu.csc216.solitaire.SomewhatSimplifiedSolitaireUI.
>
> Under the Arguments tab in the Run Configuration window, add the file name for the command line argument. Any time you need to change the command line argument, you must modify the filename in the Arguments tab. The assumption is that all files will be stored under the top level of the Eclipse project. If so, you only need to enter the file name into the Program arguments text field.

The use of the imperative mood ("create a Run Configuration" and "add the file name for the command line argument") matches the customary way of writing instructions for others as compared with instructions for yourself, which can often be done in sketchy notes using only keywords. Like the guidance on the form itself ("Describe how *you* would start *your* application." [emphasis added]), the directions in the model test report use "you," but the "you" has a different referent. It now refers to the person who will run the test, not to the student who is writing the plan: "Any time *you* need to change the command line argument, *you* must modify the filename in the Argument tab" [emphasis added].

Table for Describing Each Test. From the model test plan, the treatment group received additional guidance for describing each test in ways that would be easiest for the tester to use.

For both the control and treatment groups, three of the four columns of the table provided the same guidance, shown in Fig. 2. In first and third columns, the guidance repeats the heading and gives an additional piece of information. For the Test ID column, the students should also enter their names as

Test ID	Description	Expected Results	Actual Results
UniqueID (author)	**Preconditions:** Test description (repeatable and specific)	Expected results description (specific)	

Fig. 2. Table from the form for describing individual black box tests [15]

Test ID	Description	Expected Results	Actual Results
ZeroMinBound (author)	**Preconditions:** `StringAnalyzer` program started String? String 345 String Minimum? 0 Maximum? 7	The String contains 3 digits between 0 and 7, inclusive.	

Fig. 3. Table from Instructor's "Software Testing (CS1 & CS2" reading and "Software Testing" presentation [15-16].

authors of the tests. In the Expected Results column, the description of the expected results should be "specific." The description column is slightly more helpful. It indicates that for each test students should describe the preconditions before entering the actual description and that the test should be "repeatable and specific." For none of the columns is there guidance in the table for how to write the contents.

Both the control and treatment groups were assigned some reading that could help them describe the tests effectively. One of the two control group instructors had students read Chapter 10 from the 2nd edition of Horstmann's *Big Java* on "Testing and Debugging" [19] and used a presentation that focused on writing unit tests to introduce testing [14]. The instructor (the second author) for the other control section asked students to read "Software Testing (CS1 & CS2)," a 13-page description of various types of software testing (i.e., white-box testing and black-box testing with testing strategies). The reading packet explained that while many of the forms of testing are conducted by the programmer, the black box tests are designed by the programmer but conducted by independent testers [15]. The instructor reviewed this same material in class using a "Software Testing" presentation, a 38-slide PowerPoint that ties to the Software Testing Packet, but includes additional information about software testing [16]. Both the "Software Testing (CS1 & CS2)" packet and "Software Testing" presentation included information about what to place in each of the areas of the test plan template along with technical strategies for developing black box tests: test requirements, equivalence classes, and boundary values [15-16].

The reading included three example black box tests and showed how the test is described in the table. In effect, these samples served as the students' models for their own test cases. These same models were included in three of the 38 slides the second author used to discuss the reading in class [16]. The test cases demonstrated testing a requirement, an equivalence class, and a boundary value. All three had the same form as shown in Fig. 3, which is a test for a program, StringAnalyzer, that identifies the number of *times a digit between a minimum and maximum value, inclusive, is used in a string input.*

In this and the other examples, the second column provides only the technical information: the console interaction during test execution. For a tester, this should be enough detail for them to infer what they are supposed to do with the simple StringAnalyzer program. But the programs the students created and tested in CSC216 were more complex – all three of the projects use a graphical user interface (GUI). No help was provided in the reading or the slide deck for describing tests for GUI applications [15-16]. The reading was originally written for the introductory programming course where most programs require command line input.

The treatment group read the same "Software Testing (CS1 & CS2)" packet, including the three sample tests, and saw the same lecture presentation, in which the sample tests were repeated [15-16]. However, the treatment group also had the model test plan, which illustrated several strategies for writing their entries in "Description" and "Expected Results" columns in ways that made the tester's job much easier and

Description	Expected Results
Preconditions: The file, **`deckBad.txt`**, does exist and contains the contents as specified in the Required Files section, below.	
Enter **`deckBad.txt`** into the Program arguments text field under the Arguments tab of the Run Configurations window.	
Select the Run button.	Console output: **`Invalid file`**
	Program ends execution.

Fig. 4. Sample test from the model test plan provided to the treatment group.

demonstrated a black box test plan for a project using a GUI. See Fig. 4.

As in the Introduction of the model test plan, the text was written in the form of step-by-step directions in the imperative mood ("Enter," "Select"). Blank lines separated steps from one another to make it easier for the tester to read a step, do the step, and find the next step to read. In directions that tell the user to enter something, the text of the entry was printed in a monotype and bold font, setting it off from the rest of the directions so that it was easy for the tester to spot and read correctly. Where extra details were needed to help the tester, the detail was provided. For example, in the first step, the writer provided detailed instructions about where the tester is to enter the text: "into the Program arguments text field under the Arguments tab of the Run Configurations window."

In this example, the Expected Results column included a heading in the cell for each test: "Console Output:" followed by the text that would be displayed if the test were successful; again the expected text presented in bold monotype. Following a blank line, the next expected result was given: "Program ends execution."

Another test in the treatment group's model plan used many of the same reader-friendly techniques as illustrated in Fig. 5. The test case additionally illustrated a way to describe tests that require multiple checks on the expected results during test execution.

Description	Expected Results
Preconditions: The file, **`deck.txt`**, exists. **`deck.txt`** is specified as a command line argument and the UI has launched.	
Message: **`jzz`**	
Press Decrypt button	
Check expected output	Decrypted Text: **`abc`**
Close UI	
Check expected output	Program ends execution

Fig. 5. Sample test from the model test plan provided to the treatment group.

TABLE I. BLACK BOX TEST PLAN HIGHEST PERFORMANCE CRITERIA

Criteria	Above Average Effort
Test IDs are…	Uniquely identified and descriptive
The test description…	Fully specified with complete inputs, specific values, and preconditions
The expected results…	Are fully specified with specific output values
The test process…	Is fully provided and clear
Grammar and Spelling	Sentence structure and spelling are correct
Correct Terminology	Terminology is used correctly

Modified Grading Rubric. The grading rubric was given to students when they received the assignment. However, the grading rubric for the treatment group differed slightly, but significantly, from the grading rubric for the control group. Both rubrics had six criteria, four concerning the content of the black box test plan and two concerning its presentation. Table I presents the criteria and description of the highest (of three) levels of performance for each.

For the treatment sections, the instructor revised the rubric, leaving the descriptions of the highest levels of performance (Table I) the same, but revising the descriptions of the two lower levels (Inadequate and Adequate) to provide students a more specific understanding of what the workplace reader (the tester) would need. These revisions transform the rubric into an instructional tool to support student learning [5]. For instance, the middle level of performance for expected results changes the focus from the general characteristics of the values supplied by the student to what the tester needs in order to conduct the test.

- Control version: the expected results "are mostly fully specified, but some output values may be missing or have unspecific values."
- Treatment version: "One or two outputs missing, *but test results are clear*." [emphasis added]

The significant feature of the change is that the expected results move from generalized qualities ("missing" or "unspecified") to their impact on the tester's ability to do his or her job: determine what the test results are.

Similarly, the lowest level of performance is changed in the same way.

- Control version: the expected results "are incomplete or have no specific output values."
- Treatment version: "Missing *critical* output values." [emphasis added]

The quality of the expected values that matter again changes from generalized qualities ("missing" or "unspecific") to their specific impact on the tester: are there missing output values that are critical to the reader's ability to determine what the test results should be?

D. External Evaluation

Rating of Student Artifacts. Three CS/SE faculty members from other universities rated the student artifacts. The reviewers had over 30 years of CS/SE teaching experience in courses like CSC216 and other CS/SE courses [24].

Reviewers rated the artifacts using a rubric that contained 30 sub-criteria, grouped into ten main criteria [24]. The five-point rating scale ranged from 1 (poor) to 5 (excellent), with a sixth score of 0 or DNIA (did not include any) to be used when an element to be rated was missing from an artifact. Consistent with the experiment's goal of evaluating a way of increasing students' communication abilities without diminishing their technical abilities, the criteria were all framed from the perspective of the assignment scenario's audience: the students' workplace manager, the tester, or both. For instance, a sub-criterion under the criteria "Usability of the test descriptions" reads, "OVERALL, the TEST DESCRIPTIONS make it easy for the MANAGER to EVALUATE the tests without needing to ask for clarification." Another read, "OVERALL, the TEST DESCRIPTIONS make it easy for the TESTER to PERFORM them accurately and efficiently without needing to ask for clarification." This rubric was not shown to students, who saw only the grading rubric that is partially displayed in Table I.

Reviewers participated in a two-day training and norming session [18] conducted by Ohio's Evaluation and Assessment Center for Mathematics and Science Education [24]. The trainers flew to North Carolina to lead the training in person at a university where none of the research team worked. One-third of the time was devoted to evaluation of the selected black box test plans. The other two-thirds of the training and norming session was devoted to evaluation of other student artifacts as part of a larger project. The reviewers were introduced to the process and materials, including the rubric, they would use to evaluate the student artifacts. After evaluation of the black box test plans, inter-rater reliability was calculated and the results were shared with the evaluators. Feedback about the rubrics was provided at the end of the black box test plan evaluation session.

Inter-rater Reliability. Each student artifact was evaluated by two reviewers, which allows for assessment of Inter-rater reliability (IRR) using two-way mixed, consistency, and average-measures intra-class correlations (ICC). ICC represents the "Proportion of variance of an observation due to between-subject variability in the true scores" [22]. The measure is appropriate for the ordinal data collected by the rubric. We used the following cutoffs: < 0.40 is poor IRR; 0.40 – 0.59 is fair IRR; 0.60 – 0.74 is good IRR; and 0.75-1.0 is excellent IRR [10]. Reviewers 1 and 3 had higher IRR scores suggesting higher agreement with each other than compared to the IRR scores between Reviewers 1 and 2 and Reviewers 2 and 3. Most of the criteria, except for sub-criteria under "Style" showed fair to excellent IRR with ICCs above 0.40. Sub-criteria under the "Style" criteria showed poor or, in some cases, negative IRR. The strength of conclusions about sub-criteria under the "Style" criteria should be minimized due to poor IRR.

Internal Consistency. Internal consistency reliability for each criterion with more than one sub-criterion was computed based on factor analysis and IRR results. High internal consistency reliabilities were found for all criteria, with Cronbach's alpha values for all criteria higher than 0.80.

Analysis. Independent-sample t-tests were conducted to compare ratings according to the ten criteria in the rubric used to evaluate students' black box test plans from the treatment and control sections.

V. RESULTS

Table II shows the ratings of artifacts from control (non-RITCI) and treatment (RITCI) versions of CSC 216. Students in the treatment sections performed significantly better ($p<0.05$) than their control section peers on black box test plan assignments on six of the ten criteria: "Introduction," "Usability," "Comprehensibility," "Accessibility," "Style," and "Overall Effectiveness." For four of the six criteria, the p value was 0.001 or less. For no criterion did the control group perform significantly better than the treatment group.

For the first nine criteria, Table II reports on the raters' perceptions of student's performance with respect to specific criteria for writing black box test plans. By presenting the results for the four subcategories of the tenth criterion ("Overall"), Table III presents the rater's holistic impression of the students' technical and communication abilities from the perspectives of the manager and the tester. The raters judged that the students in the treatment group outperformed the students in the control group in all subcategories at $p<0.003$.

VI. DISCUSSION AND IMPLICATIONS

The results in Tables II and III suggest some tantalizing responses to our two research questions.

Table II indicates that for this assignment and student sample, students can learn to perform better at writing certain elements of a workplace communication from the perspectives of two kinds of readers—managers and testers—using methods that require no additional class time devoted to teaching communication, and therefore without time taken away from technical instruction. For six of the elements on the rubric, generated by CS/SE specialists with the help of experts in other fields, the writing of the students in the treatment group received significantly higher ratings than the writing of students in the control group. For all but one of the remaining four criteria, the mean of the ratings for the treatment exceeded the mean of the ratings for the control. The exception is coverage, and the means are within 0.06. One item to note is that the differences in instruction of the technical topic of testing in the control sections likely had little effect on the differences we see in Table II. The four criteria without significant differences between control and treatment are the criteria most closely associated with assessing technical knowledge: coverage of tests over the problem statement, test IDs, expected results, and actual results. The only technical

TABLE II. Comparison of ratings for the ten black box test plan criteria from the control and treatment groups

Rubric Item	Group	n	M	SD	t	df	p
Introduction	Control	29	2.62	1.1	-0.209	41	**0.043**
	Treatment	14	3.49	1.4			
Coverage	Control	31	4.03	0.8	0.36	43	0.719
	Treatment	14	3.93	0.8			
Test IDs	Control	31	4.08	1.1	-1.02	43	0.312
	Treatment	14	4.43	0.8			
Usability of the test descriptions	Control	31	3.09	0.8	-3.72	43	**0.001**
	Treatment	14	4.09	0.6			
Expected Results	Control	31	3.68	1.0	-1.91	43	0.062
	Treatment	14	4.25	0.6			
Actual Results	Control	31	3.98	0.8	-1.23	43	0.224
	Treatment	14	4.32	0.8			
Compre-hensibility	Control	31	3.90	0.7	-3.76	41	**0.010**
	Treatment	14	4.10	0.5			
Accessibility	Control	31	3.9	0.7	-3.76	41	**0.001**
	Treatment	14	4.52	0.4			
Style	Control	31	4.01	0.4	-6.22	36	**< 0.001**
	Treatment	14	4.73	0.3			
Overall Effect-iveness	Control	31	3.09	0.7	-4.82	43	**< 0.001**
	Treatment	14	4.13	0.5			

TABLE III. Comparison of ratings for the subcategories of the Overall criterion from the control and treatment groups.

Rubric Item	Group	n	M	SD	t	df	p
Manager Evaluation [a]	Control	31	3.35	0.7	-6.12	34	**< 0.001**
	Treatment	14	4.46	0.5			
Tester Conduct [b]	Control	31	2.82	0.9	-5.43	43	**< 0.001**
	Treatment	14	4.25	0.6			
Skilled CS/SE [c]	Control	31	3.24	0.6	-3.10	43	**0.003**
	Treatment	14	3.93	0.8			
Skilled Communi-cator[d]	Control	31	2.94	0.8	-3.57	43	**0.001**
	Treatment	14	3.86	0.7			

[a] OVERALL, the communication will ENABLE the MANAGER to EVALUATE the test plan EFFICIENTLY

[b] OVERALL, the communication will enable the TESTER to CONDUCT the test EFFICIENTLY and ACCURATELY

[c] OVERALL, the communication will PERSUADE the MANAGER and the TESTER that the writer is a SKILLED COMPUTER SCIENTIST / SOFTWARE ENGINEER

[d] OVERALL, the communication will PERSUADE the MANAGER and the TESTER that the writer is a SKILLED COMMUNICATOR

criterion with a significant difference is the usability of the test descriptions, for which the treatment group's rating is higher.

The primary implication of our study is that using Reader-Oriented Integrated Technical and Communication Instruction (RITCI) without using class time deserves more research. Some of the questions worth investigating relate to the specifics of our experiment. Would similar results be obtained with larger samples at other institutions or with other assignments? If so, are there ways to improve our approach? For instance, the model black box test plan did not include any annotations to highlight and explain the writing strategies made to create an effective communication. Could the addition of annotations that highlight and explain the strategies used produce greater improvement? What about revising the assignment's instructions to place greater emphasis on meeting the needs of the manager and tester and revising the grading rubric to emphasize more explicitly the impact of the various levels of performance on these readers? Would drawing students' attention to the importance of the criteria where there was not a statistically significant improvement in our experiment, such as the precision of the expected and actual results, also produce positive results?

Second, could additional gains in student learning be obtained using other strategies advocated by writing-in-the disciplines specialists? These include preparation of more than one draft and peer review [1]. What about user testing, in which students swap drafts and test one another's plans, a technique used frequently in technical writing courses? Various programs exist that enable faculty to assign and monitor these activities as homework, so that class time can remain focused on technical content. Students submit their drafts online, the program distributes the drafts for review. Some of these programs handle grading the reviews automatically, requiring little instructor time beyond setting the variables.

A third line of research could investigate the extent (and limits?) of using no-class-time approaches to teaching writing in a CS/SE program. Specialists in writing-across-the-disciplines assume that explicit, in-class instruction is an essential element of the writing instruction provided in disciplinary classes [1]. Would students in the treatment group have performed even better if they had received some instruction in class? Would in-class instruction be necessary to develop student writing for the more complex communication assignments that would occur later in a CS/SE program? Could writing instruction be facilitated as part of an inverted or flipped classroom?

Fourth, are the writing abilities students learned in the treatment sections transferrable to other courses—and are there ways to foster that transfer without diminishing attention to technical course content? The writing tasks for which the treatment group demonstrated significant improvement—writing usable directions and preparing comprehensible, accessible, communications in a style their readers will think appropriate—are equally important to writing success in many other professional writing situations.

It is worth noting that the use of a rubric like the one given to students in our experiment requires no special training in writing instruction for the CS/SE instructor or teaching assistant. To evaluate students' performance, the instructor or teaching assistants need only appraise each student's communication from the perspective of the manager and tester, not that of the English or CS/SE teacher.

In sum, our research suggests that it is possible for CS/SE instructors to increase students domain-specific writing abilities without special knowledge of writing pedagogy and without diminishing the attention given to a course's technical content, at least in introductory-level courses with software engineering content.

A replication package is available from the second author by request. It consists of the "Software Testing (CS1 & CS2)" packet [15], "Software Testing" presentation [16], the evaluation rubrics [24], assignments, and model documents.

VII. Threats to Validity

Empirical education research inherently has many threats to validity due to the nature of research in a classroom [3, 13].

A. Internal Validity

To increase the internal validity of our study, we wanted to minimize bias in the experiment. One of the factors is selection bias. Students choose when to enroll in the course and had the opportunity to opt out of the study. We minimized one impact of selection bias by using a fall semester offering of CSC216 for both the control and treatment groups. The fall offering of CSC216 contains students who are typically in a single cohort for the standard undergraduate curriculum.

The control and treatment sections of the course differed in several ways in addition to the intervention factors identified as the treatment (revision of the assignment and rubric and creation of the model student test plan). One major difference is that members of the control group were paired with a different partner for all three projects. For the treatment section, students completed projects one and two individually and project three in a pair. The results suggest that a RITCI course shows communication and technical learning gains by students even when assignments are completed individually. Additional research comparing individual and paired work in a RITCI course could further inform the research.

The programs for which the students were to design and prepare black box test plans differed. For the treatment group, they were more complex, which may have increased students' engagement with their projects, which could have increased their learning. Finally, while no time was devoted to teaching the students how to write effectively, the 13-page reading and the PowerPoint review in class both introduced students to the fact that in the profession, the creators of black box test plans communicate the plans to others who use the communication to conduct the test. Without understanding of the critical role of communication in the CS/SE profession, students may have little motivation for developing communication skills—motivation that was provided in our experiment.

We did not consider confounding factors like GPA, grades in prerequisite courses, or other demographic information when comparing the control and treatment groups. The three sections taught by the second author (one control, both treatment) had a common 100-point final exam. Both treatment sections had a median almost six points higher than the control section. This may mean that students in the treatment were stronger students or these data may suggest that the treatment interventions (beyond just the black box test plan discussed in this paper) contributed to increased technical understanding.

The second author taught three of the four sections studied. The other instructor used slightly different materials for the reading and associated lecture that emphasized white-box testing over black box test plans [14, 19]. The changes to the assignments and rubrics between semesters are the result of workshops and discussions with communication across the curriculum experts about how best to create assignments that foster legitimate peripheral participation for CS/SE students. To reduce experimenter bias, the artifacts were evaluated by external reviewers using a different rubric and data were analyzed by an external evaluator.

The testing scenario described in "Software Testing (CS1 & CS2)" where a programmer writes the black box test plan before development begins and then the tester runs the tests after the program is complete is not the only scenario when working with black box test plans. Organizations may have the tester write the black box test plans rather than the developer.

B. External Validity

Externally valid studies can generalize to other setting beyond the specifics of the study. While we hypothesize that our results will generalize to other courses and other institutions, we cannot currently make that claim. The Fall 2013 offering of CSC216 taught by the second author in a similar manner to the Fall 2012 offering had a final exam median almost one point above the Fall 2011 control median. The Fall 2014 offerings of the course had a final exam median 10 points higher than the Fall 2011 control median, but that is likely due to another intervention during that offering. All medians are close enough to suggest that the incorporation of communication does no harm to students' technical skills. Further assessment of external reviewers on other treatment offerings of CSC216 and for similar external classes are required to increase confidence that minimal instruction by CS/SE educators is required for students to experience communication learning gains.

Another threat to external validity is that the number of artifacts rated for the treatment group is small. Future work can replicate the study on additional students at NC State or other institutions to increase generalizability of the results.

C. Construct Validity

To maximize construct validity, we carefully constructed, reviewed, and revised the rubric developed for use by the external reviewers. The rubric was reviewed by a team of CS/SE specialists, technical communication specialists, and measurements specialists at Ohio's Evaluation and Assessment Center for Mathematics and Science Education [24].

VIII. Conclusions

The overall result of the differences shown in Table III is that the treatment groups' artifacts were superior to the control groups' artifacts at "enabl[ing] the manager to evaluate the test plan efficiently" and at "enabl[ing] the tester to conduct the test efficiently and accurately." These differences also impacted the rater's judgment of the communication competence of the student writers: the raters indicated that the treatment group's artifacts were more likely to "persuade the manager and the tester that the writer was a skilled communicator." Taken together these results suggest a confirmation of our first research hypothesis: Using RITCI in a CS course without taking class time away from technical instruction can increase the quality of students' communication abilities.

Tables II and III suggest a similar confirmation of our second hypothesis: rather than diminishing students' technical abilities, as some have feared, the use of RITCI in a CS course without taking class time away from technical instruction can increase the quality of students' technical abilities. Students score higher on the first nine criteria only if they demonstrate that they have understood and can apply the technical dimensions of writing black box test cases. Table II shows that the raters judged that the treatment group understood and applied the knowledge related to five of the nine specific criteria more thoroughly and skillfully than the control group, with the caveat that one of the criteria, "Style", had a poor IRR. Table III shows that the raters judged that the reports written by the treatment group were significantly more likely than the reports written by the control group to persuade the manager and tester that the student was a skilled computer scientist/software engineer. Thus, it appears that the increase in communication abilities by the treatment group was not accompanied by a reduction in technical knowledge and skills. Taken together these results suggest the opposite. The response to our second research question seems to be that using RITCI in a CS course without taking class time away from technical instruction can increase students' technical abilities.

ACKNOWLEDGMENT

We thank the other CSC216 Instructors, other members of the CPATH team, external reviewers, Ohio's Evaluation and Assessment Center for Mathematics and Science Education, and Yue Li. This work was funded by NSF CPATH II Award CCF-0939122 and CCF-0939081. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

REFERENCES

[1] J. Bean, *Engaging Ideas: The Professor's Guide to Integrating Writing, Critical Thinking, and Active Learning in the Classroom*, San Francisco, CA, Jossey-Bass, 2011.

[2] A. Begal and B. Simon, "Novice software developers, all over again," Int. Computing Education Research Conf., Sydney, Australia, September 6-7, 2008, pp. 3-14.

[3] C. Bishop-Clark and B. Dietz-Uhler, *Engaging in the Scholarship of Teaching and Learning*, Sterling, VA, Stylus Publishing, 2012.

[4] British Computer Society Specialist Interest Group in Software Testing, "Standard for software component testing," Working Draft 3.4, April 27, 2001 http://www.testingstandards.co.uk/Component%20Testing.pdf.

[5] J. E. Burge, G. C. Gannod, P. V. Anderson, K. Rosine, M. A. Vouk, M. Carter, "Characterizing communication instruction in computer science and engineering programs: methods and applications," Frontiers in Education Conference, Seattle, WA, October 3-6, 2012, pp. 1-6.

[6] J. E. Burge, G. C. Gannod, M. Carter, A. Howard, B. Schultz, M. Vouk, D. Wright, P. Anderson, "Developing CS/SE students' communication abilities through a program-wide framework," Technical Symposium on Computer Science Education '14, Atlanta, GA, March 5-8, 2014, pp. 579-584.

[7] M. Carter, "Ways of knowing, doing, and writing in the disciplines," *College Composition and Communication,* vol. 58, no.3, 2007, pp. 385-418.

[8] M. Carter, R. Fornaro, S. Heckman, M. Heil, "Creating a progression of writing, speaking, & teaming learning Outcomes in undergraduate computer science/software engineering curricula," World Engineering Education Forum (WEEF), Buenos Aires, Argentina, October 15-18, 2012.

[9] M. Carter, M. Vouk, G. Gannod, J. Burge, P. Anderson, M. Hoffman, "Communication genres: integrating communication into the software engineering curriculum," Conf. on Software Engineering Education & Training 2011, pp. 21-30

[10] D. V. Cicchetti, "Guidelines, criteria, and rules of thumb for evaluating normed and standardized assessment instruments in psychology," *Psychological Assessment*, vol. 6, no. 4, 1994, pp. 284-290.

[11] R. Dugan and V. Polanski, "Writing for computer science: a taxonomy of writing tasks and general advice," *Journal of Computing Sciences in Colleges*, vol. 21, no. 6, June 2006, pp. 191-203.

[12] K. Falkner and N. J. G. Falkner, "Integrating communication skills into the computer science curriculum," SIGCSE'12, February 29-March3, 2012, pp. 379-384.

[13] S. Fincher and M. Petre, eds., *Computer Science Education Research*, Lisse, The Netherlands, Taylor & Francis, 2004.

[14] E. Gehringer, "CSC 216: Lecture 3," NC State University, August 25, 2011, http://www.csc.ncsu.edu/faculty/efg/216/f11/www/lectures/notes/lec3.ppt.

[15] S. Heckman, "Software testing (CS1 & CS2)," NC State University, August, 6, 2010, http://courses.ncsu.edu/csc116/common/TestingMaterials/Testing_SSH_v4.pdf.

[16] S. Heckman, "Software testing," NC State University, September 1, 2013, http://www.csc.ncsu.edu/courses/csc216-common/Heckman/lectures/04_Testing.pdf.

[17] M. E. Hoffman, P. V. Anderson, M. Gustafsson, "Workplace scenarios to integrate communication skills and content: a case study," SIGCSE' 14, Atlanta, GA, March 5-8, 2014, pp. 349-354.

[18] C. Holmes and M. Oakleaf, "The official (and unofficial) rules for norming rubrics successfully," *Journal of Academic Librarianship*, vol. 39, no. 6, 2013, pp. 599-602.

[19] C. Horstmann, *Big Java*, Hoboken, NJ, John Wiley & Sons, Inc., 2006.

[20] A. Karatsolis, I. Cervesato, N. Abu-Ghazaleh, Y. Cooper, K. Harras, K. Oflazer, and T. Sans, "Getting CS undergraduates to communicate effectively," Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education, Darmstadt, Germany, June 27-29, 2011, pp.283-287.

[21] J. Lave and E. Wenger, *Situated Learning: Legitimate Peripheral Participation*, Cambridge University Press, 1991.

[22] J. Ludbrook, "Confidence in Altman-Bland Plots: A Critical Review of the Method of Differences," *Clinical and Experimental Pharmacology and Physiology*, vol. 37, no. 2, 2010, pp. 143-149.

[23] L. Reave, "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, p. 452-490.

[24] S. B. Woodruff, L. Yue, E. E. Ryan, "Evaluation of CPATH II: Incorporating Communication Outcomes into the Computer Science Curriculum, Final Technical Report," Miami University, Ohio's Evaluation & Assessment Center for Mathematics and Science Education, Oxford, OH, March 2014.