A Preliminary Review of Undergraduate Programming Students' Perspectives on Writing Tests, Working with Others, & Using Peer Testing

Alessio Gaspar, Sarah Langevin
University of South Florida in Lakeland
3433 Winter Lake Road
Lakeland, FL 33803-9807
863-667-7088
alessio@usf.edu

Naomi Boyer
Polk State College
999 Ave. H. NE.
Winter Haven, Fl. 33881
863-298-6854
nboyer@polk.edu

Ralph Tindell
University of South Florida
4202 E. Fowler Avenue
Tampa, FL 33620
863-914-3032
rtindell@cse.usf.edu

CEReAL group – http://cereal.forest.usf.edu/

ABSTRACT

Techniques such as Pair Programming, or allowing students to run their programs against a reference test harness, have demonstrated their effectiveness in improving grades or retention rates. This paper proposes to supplement the existing literature by investigating students' perceptions of the benefits of writing tests, working with other students and using Peer Testing. Responses to an online anonymous survey cast new light on the relation between testing and programming and confirm previously postulated limitations of collaborative approaches; i.e. the unbalanced nature of contributions and lack of didactic interactions in student groups. We then examine how Peer Testing is perceived and discuss its relation to both collaboration and test-based pedagogies.

Categories and Subject Descriptors

K.3.2 [**Computers and Education]**: Computer and Information Science Education – *computer science education, curriculum*.

General Terms

Measurement, Human Factors.

Keywords

Peer Testing, Novice Programmers, Programming Pedagogy.

1. INTRODUCTION

A significant body of literature exists on the benefits of having novice programmers write tests; refer to **[2][3]** for examples. Using testing principles to enable students to run their programs against the instructor's reference test harness has also proven helpful **[4][5]**. Peer Testing, a variant in which students run their programs against other students' tests, has been less studied **[6][7]**. This idea is based on the assumption, well-grounded in educational theory, that students are more likely to be challenged by tests that are within their zone of proximal development **[13]** than by tests provided by the instructor. Previous work also highlights the relation between Peer Testing and the introduction of constructivism in pair programming activities **[8][9][10][11]**.

This paper proposes to investigate in more details the specific benefits to students of writing tests, working with others freely, and using Peer Testing. To gain deeper insight, we identify subpopulations of respondents characterized by their affinity for writing tests or working with others. These subpopulations are then compared to one another with respect to attitudes toward all three learning activities.

The remainder of this paper is organized as follows. Section 2 provides background information on this study and details on our methodology. Sections 3, 4 and 5, respectively, analyze students' perspectives on using tests, working with others and using Peer Testing. Section 6 discusses findings and future work.

2. BACKGROUND

This section establishes our study's specifics in terms of the survey population, material taught, and pedagogies used. We also discuss the research methodology for collecting data.

2.1 Research Methodology

An anonymous online survey hosted on Survey Monkey was used to gather students' attitudes and perspectives. A link for the survey was provided to students via announcement on the Learning Management System (Blackboard).

Participation was optional but earned extra credit to reward participants for their time. To keep the survey anonymous, a "key" was provided on the last page of the survey. Students were invited to email that key to their instructor for extra points. To discourage students from providing random responses to get the key, an option was available to obtain the key without responding to the survey.

Most questions allowed respondents to provide feedback using one of the following Likert scales:

- **5-point agreement Likert scale** with labels "Strongly Disagree", "Disagree", "Neutral", "Agree", "Strongly Disagree". Results are presented in terms of labels "Agree" / "Neutral" / "Disagree" by aggregating responses on the 2 first and 2 last labels of the original scale. Sample questions: Q3, Q4, Q5, Q6, Q8, Q12, Q13, Q14, Q15, Q16.
- **5-point frequency Likert scale** with labels "Almost Never", "Seldom", "Sometimes", "Often", "Most of the time". Responses were presented using labels "Rarely" / "Sometimes" / "Often" by aggregating responses on the two first and two last labels of the original scale. Sample questions: Q9, Q10.

- **3-point frequency Likert scale** with labels “Often”, “Sometimes”, “Never”. Sample question: Q7

In addition, average ratings were reported by assigning integer values starting at 1 to each label.

2.2 Surveyed Population

The survey population consisted of junior standing undergraduates, enrolled in IT Program Design during fall 2012. A total of 30 students visited the survey page, 14 opted to obtain the key without responding, 16 opted to answer the questions but only 15 of them actually answered the questions.

IT Program Design is the 2nd programming course for IT majors. To gain admittance, tudents must pass an introductory programming class, which is often taught in Java with a focus on a “fundamentals first”, as opposed to an “objects first” pedagogy. IT Program Design focusses on three main learning outcomes.

First, it explicitly teaches the thought process used to deliver programs from requirements. Many students learn in other courses only to work with requirements that fully specify solutions in plain English, which they then translate into Java.

Second, it introduces students to system-level concepts such as execution stacks, pointers, memory allocation.

Third, it prepares students for upper-level system-oriented offerings using the C language **[12]**.

Topics covered map to **[1]**: fundamentals, functions, arrays, pointers, strings, dynamic memory allocation, user-defined types and elementary data structures. These topics are covered in 7 modules. The material, excluding graded assignments and quizzes, is available at

http://cereal.forest.usf.edu/clue/progdesign/.

2.3 Pedagogy

IT Program Design is taught as an online asynchronous offering. Students work on the various activities in each module on their own schedule as long as they meet set deadlines.

Each module is two weeks long. The first week is devoted to reading assignments; posting questions on forums; meeting on Elluminate to get personalized help; working on apprenticeship exercises; and watching step-by-step solution videos. The latter are implementations of cognitive apprenticeship **[8]**. By the end of the each module’s first week students take a graded quiz.

During the second week, students work on a mini project. Detailed feedback is provided along with a “satisfactory or not” evaluation, which earns students participation points.

IT Program Design’s main pedagogical innovations reside in its use of apprenticeship exercises, testing and peer testing.

The topic of apprenticeship exercises is out of the scope of this paper, but the reader is referred to **[8][9][10][11]** for details. Writing a test harnesss is strongly encouraged for non-graded practice assignments (PAs) and required in graded programming assignments (GPAs). Originally, testing was introduced to help address the problem of loss of intentionality in programming **[10]**. Peer Testing was used in all practice and graded assignments.

3. Perspectives on Testing

This section investigates students’ perspectives on writing tests for their programs as part of learning to program.

3.1 Survey Questions

Our first question, Q2, focused on establishing the degree to which students used tests out of compliance vs. genuinely appreciating their benefits. The question was as follows:

Q2 *Which of the following describes best your usage of tests this past semester;*

1. *I wrote tests for all my programs, graded or not*
2. *I only wrote tests for graded programming assignments, even when they didn't require it*
3. *I only wrote tests when the graded programming assignments required it*

Students selected one of the three available options. The first one captures positive attitudes toward testing whereby the respondent used tests in all the programs they wrote without external reward. The second is meant to identify respondents who value testing enough to leverage it in graded assignments but who may not see the point in using it in every program they write. The last statement is meant for respondents who only used tests in order not to lose points. This question allowed us to identify two sub-populations:

P1 *Students who adopted testing in all their activities without need for external rewards – Q2 response #1*

P2 *Students who used testing as compliance with mandatory requirements – Q2 response #3*

The next questions, Q3 / Q4 / Q5, evaluated the perceived usefulness of using tests for, respectively, debugging programs, understanding requirements and improving programming skills.

Q3 *Designing tests helped me find errors in my programs.*

Q4 *Designing tests helped me better understand the requirements for my programs*

Q5 *Designing tests helped me learn to program.*

Students were invited to rate their agreement with the above statements using a 5-points agreement Likert scale.

3.2 Observations

Table 1 summarizes responses to Q2. A single student adopted testing only for graded assignments. The majority of others adopted testing for all their programs; thus, suggesting its perceived usefulness. This assessment is in line with other studies on the positive impact of testing e.g. **[2][3][4][5]**.

Table 1 – Q2

Response	**#**
I wrote tests for all my programs, graded or not	9
I only wrote tests for graded programming assignments, even when they didn't require it	1
I only wrote tests when the graded programming assignments required it	5

Table 1 shows that the sizes of subpopulations P1 and P2 are, respectively, N=9 and N=5. **Table 2** shows the levels of agreement of populations P0, P1 and P2 with questions Q3 to Q5. A majority of respondents agreed to the three types of benefits.

Table 2 – Q3 to Q5

Question	**P**	**Disagree**	**Neutral**	**Agree**	**Avg**
Q3	P0	1	5	9	3.87
	P1	0	2	7	4.22
	P2	1	3	1	3.00
Q4	P0	3	4	8	3.60
	P1	0	2	7	4.22
	P2	3	2	0	2.40
Q5	P0	3	3	9	3.53
	P1	0	1	8	4.22
	P2	3	2	0	2.20

Looking at the average rating for P0 instead of the aggregated responses suggests a marginally stronger agreement level for Q3, with Q4 and then Q5. Benefits to students are most obvious as they relate to finding bugs rather than understanding requirements or even learning to program in general. P2 respondents follow the same ranking while P1 respondents equally rate their agreement to all three benefits.

Results suggest that students who do not find a benefit in writing tests also only use tests when required to do so.

3.3 Discussions

The initial impression is that P2 students seem to be compliant learners in so far as they used tests only when required to do so in graded assignments. However, most compliant learners should have seized the option described in **section 2.3,** which allowed them to get the survey key without taking the survey.

It is therefore possible, in contradiction to the available literature **[2][3][4][5][9]**, that writing tests was perceived as genuinely useless by P2 students. **Table 2** shows that disagreement levels for P2 respondents are non-uniform. The highest agreement level for P2 was on that writing tests helped in finding bugs. Due to the overall agreement, regardless of subpopulations, this outcome is potentially one of the main agreed-upon benefits of writing tests.

Writing tests to formalize requirements was expected to be helpful by triggering questions that would help students better understand expectations. However, P2 responses suggest that representing requirements in tests was not more helpful than simply implementing them directly in the program. This contrasts the educational benefits of writing tests with those observed by software engineers for whom they represent a more objective formalization of requirements.

Similarly, P2 students felt that writing tests did not help improve programming skills in general. While testing undeniably yields better quality software, the skills involved in writing tests are not necessarily the same as those used in writing programs. As such, tests might help students become better developers in the long term without supporting in the short term their acquisition of more elementary programming skills.

4. Perspective on Programming with others

This section establishes the attitude of our students regarding working with others on programming tasks.

4.1 Survey Questions

The first question is meant to identify subpopulations based on whether respondents worked with others.

Q7 *Rate how often you've worked on programming tasks with other students regardless of whether they were graded or non-graded; e.g. programming assignments, group projects, exercises or simply while participating in "study groups."*

Respondents were asked to answer the question two times: "*Before taking this course*" and "*During this course*". Each response used a 3-point Frequency Likert scale. This question allowed us to distinguish two sub-populations:

P3 *Students who actually worked with other students during this course; Q7 responses "Often" or "Sometimes".*

P4 *Students who never worked with other students in this offering; Q7 responses "Never".*

Question Q8 used a 5-point agreement Likert scale to measure respondents' perspectives on working with other students.

Q8 *Rate your levels of agreement with the following statements. Working with other students on programming tasks...*

- *…is more enjoyable than working alone*
- *... is more beneficial to my grades than working alone*
- *... is more beneficial to improving my individual programming skills than working alone*
- *... is more beneficial to understanding how to apply the lectures than working alone*

Question Q9 used a 5-point frequency Likert scale to measure how often students "take the lead" in terms of efforts or tutoring.

Q9 *How "balanced" are the contributions of the other students working with you?*

- *I end up contributing more toward the end result than others*
- *I end up explaining more to others than they explain to me*

Q10 offered a list of activities for respondents to rate using the same 5-point frequency Likert scale. These activities exemplified leadership (1), responsibility (2), tutoring by explanations (3,4), involvement in others' work (5,6), tutoring by lecturing (7) and constructivist tutoring (8). See **Table 6** for activities list.

Q10 *Rate how much of the following types of contributions you provide when working with other students on programming tasks <list of contributions follows – see **Table 6** for details>*

4.2 Observations

Table 3 suggests an increase in how often students worked with others from previous semesters and this offering. It also establishes the size of our subpopulations; P3 is N=9, P4 is N=6.

Table 3 – Q7

Responses	Never	Some times	Often	Avg
Before this offering	8	7	0	2.53
During this offering	6	5	4	2.13

Table 4 shows the response distribution for Q8. When looking at all respondents, P0, students agreed in majority to all statements with marginal differences. The average ratings show statements #2 and #3 as being in the lead. The fact that students rated identically the benefits to their grades and to the improvement of their programming skills suggests they perceive both as equivalent.

When comparing sub-populations P4 & P3 students from the former have more pronounced disagreement / neutral feedback.

Table 4 – Q8

Responses		P	Disagree	Neutral	Agree	Avg
1	…is more enjoyable than working alone	P0 P3 P4	2 0 2	5 2 3	8 7 1	3.47 4.11 2.50
2	... is more beneficial to my grades than working alone	P0 P3 P4	2 0 2	4 1 3	9 8 1	3.60 4.22 2.67
3	... is more beneficial to improving my individual programming skills than working alone	P0 P3 P4	2 0 2	4 1 3	9 8 1	3.60 4.22 2.67
4	... is more beneficial to understanding how to apply the lectures than working alone	P0 P3 P4	3 1 2	5 2 3	7 6 1	3.40 3.89 2.67

Table 5 shows Q9 responses. Most students rarely take the lead in contributing and even more rarely in explaining to others. This

suggests that, as could be expected, regardless of whether students work together in a balanced manner **[9]** their focus is not on helping the other students but on completing the project.

Table 5 – Q9

Responses	P	Rarely	Some times	Often	Avg
I end up contributing more toward the end result than others	P0 P3 P4	7 4 3	4 3 1	4 2 2	2.67
I end up explaining more to others than they explain to me	P0 P3 P4	8 5 3	3 2 2	4 2 1	2.53 2.44 2.67

Table 6 Q10 responses suggest that the majority of respondents rarely engage in the activities we listed when working with others.

Table 6 – Q10

#	Statement	P	Rarely	Some times	Often	Avg
1	Leading by breaking down the problem then assigning tasks to others & myself	P0 P3 P4	8 5 3	6 3 3	1 1 0	2.07 2.11 2.00
2	Implementing the parts of the overall project which were assigned to me	P0 P3 P4	5 3 2	5 2 3	5 4 1	2.87 3.00 2.67
3	Explaining the parts I implemented to the other students	P0 P3 P4	4 2 2	10 6 4	1 1 0	2.60 2.78 2.33
4	Explaining to other students how to implement their parts	P0 P3 P4	7 4 3	7 4 3	1 1 0	2.27 2.33 2.17
5	Fixing bugs in other students' parts	P0 P3 P4	10 7 3	5 2 3	0	1.93 1.78 2.17
6	Explain their bugs to other students	P0 P3 P4	9 6 3	6 3 3	0	2.07 2.00 2.17
7	Helping others improve their programming skills by "lecturing them" or providing advice	P0 P3 P4	8 5 3	7 4 3	0	2.00
8	Helping others improve their programming skills by understanding what their misconceptions are then providing counter examples	P0 P3 P4	9 5 4	5 3 2	1 1 0	2.07 2.22 1.83

Let us look at these responses based on the attitudes they reflect;

Leadership (1) – This activity implies an ability or willingness to organize programming tasks, whether alone or in a group. It was rarely the focus of respondents, regardless of the population. This is not surprising in so far that novice programmers should not be expected to easily take on the role of "developer lead".

Responsibility (2) – Regardless of the subpopulation considered, respondents are split on how often they implement their own parts. This suggests unbalanced contributions in teams.

Instructivist Tutoring (3,4,7) – While respondents were likely to explain what they did to others (3), they were not frequently involved in "explaining" to others how to do their work (4) and even less in tutoring them (7). This suggests that, as was hypothesized before **[8][9]**, the primary goal of students working together on programming projects is to complete the work rather than to help each other improve skills.

Constructivist Tutoring (5,6,8) – Beyond the willingness or ability to help other students, responses suggest an even less frequent involvement in activities which require understanding a partner's mistakes or misconceptions. As partners in a project, students probably perceive investing in understanding the work or thoughts of others as a waste of their time. This is in contrast with instructors' constructivist beliefs that put such understanding of students' mistakes or previous knowledge at the forefront of effective teaching.

4.3 Discussion

The consensus in computing education literature is that collaboration among students, e.g. pair programming, is efficient for individual skills development and retention **[16]**. However, previous work led to hypothesizing that the nature of the student-to-student interaction in such collaborations might be improved **[8][9]**. More specifically, the unbalanced contributions of students along with the lack of didactic dialog suggest that techniques such as Peer Testing might be an improvement.

While professional programming teams often follow the mentor / apprentice variant **[3]**, each member has already proven his/her ability to program individually. In students groups, differences in skill levels are likely to be much more pronounced. **Table 5** confirms that most students see themselves as infrequently contributing more than others. Similarly, **Table 6** shows, via responses to item #2, that students are equally split in the frequency they implement the tasks they were assigned. Respondents reported working with others, i.e., P3, seem to report implementing their assigned tasks more often than those who did not work with others, i.e., P4.

Similarly, **Table 6** item #1 suggests that few students, regardless of subpopulation, frequently take the lead. This item also captures the ability to break down the task at hand into smaller sub-problems, which is not expected to be wide-spread among novice programmers. These results are consistent with expectations.

Therefore, responses confirm that groups are based on unbalanced contributions: some students contribute more often, others do so sometimes, but the majority lack involvement.

Further, our data provide insight regarding the nature of the didactic exchanges taking place inside such groups. **Table 6** reveals that students are even less likely to offer educational help to others than to contribute a fair amount of work. While a good proportion of students are willing to explain, few are interested in lecturing others and even fewer in getting involved with other students' work. This is in stark contrast with the computing education researchers' focus on constructivism vs. instructivism **[14][15]**. Constructivism requires instructors to "get involved" with students' attempts, understand their misconceptions and build on their previous knowledge, rather than simply state the solution. When students work together, instructivism prevails, thus limiting potential educational benefits.

Together, these observations confirm the two above-mentioned hypotheses that motivated the design of Peer Testing **[8][9]**. The next section will examine whether students' perception of Peer

Testing is in alignment with its expected potential to address the shortcomings confirmed in this section.

5. Perspectives on Peer Testing

This section explores students' perspective on Peer Testing.

5.1 Surveys Questions

The first questions, Q12 / Q13 / Q14, were meant to explore how Peer Testing supported students' learning.

Q12 *Being able to use my classmates' tests helped me improve my own tests.*

Q13 *Being able to use my classmates' tests helped me improve my own programs by identifying missing features in them.*

Q14 *Being able to use my classmates' tests helped me improve my own programs by finding errors in them.*

The next question, Q15, inquired as to whether Peer Testing was successful in allowing students to receive help from others while still requiring them to understand and fix their own bugs.

Q15 *Using classmates' tests forced me to figure out my errors myself instead of letting a classmate do it for me.*

The next question, Q16, went one step further by asking whether students saw this approach as more beneficial to improving their skills than just having someone else fix their bugs.

Q16 *This form of collaboration led me to develop my own programming skills more than if I had only shared programs directly with classmates.*

The last question, Q17, was meant to capture whether the students' experience with Peer Testing was positive.

Q17 *In your next programming-related offering would you like to be offered the option to use peer testing again?*

5.2 Observations

Table 7 summarizes responses for the various subpopulations identified so far. We will focus on P0 first.

The questions aimed at establishing Peer Testing's usefulness, i.e., Q12, Q13, Q14, feature high levels of agreement. This suggests that students perceive benefits from exchanging tests with others not only to improve their own tests, obviously, but also to improve their programs. These perceived benefits are paired with an overall positive experience as illustrated by a majority of students expressing agreement to Q17.

Table 7 – Q12 to Q17with P0

Question	P	Disagree	Neutral	Agree	Avg
Q12	P0	2	1	12	4.07
	P1	1	0	8	4.22
	P2	1	1	3	3.60
	P3	0	1	8	4.33
	P4	2	0	4	3.67
Q13	P0	3	0	12	3.93
	P1	1	0	8	4.11
	P2	2	0	3	3.40
	P3	1	0	8	4.22
	P4	2	0	4	3.50
Q14	P0	3	1	11	3.87
	P1	1	1	7	4.00
	P2	2	0	3	3.40
	P3	0	0	9	4.44
	P4	3	1	2	3.00
Q15	P0	3	2	10	3.73
	P1	2	0	7	3.78
	P2	1	2	2	3.40
	P3	1	1	7	4.00
	P4	2	1	3	3.33
Q16	P0	3	5	7	3.40
	P1	1	3	5	3.56
	P2	2	2	1	2.80
	P3	1	3	5	3.67
	P4	2	2	2	3.00
Q17	P0	0	5	10	n/a
	P1	0	2	7	
	P2	0	3	1	
	P3	0	2	7	
	P4	0	3	3	

However, students are a bit more divided regarding the idea that Peer Testing is more beneficial than sharing programs; see Q16.

5.3 Relation to group work predisposition

Understanding the general perspective of respondents about working with others is essential to making sense of their attitude toward Peer Testing. **Table 7** responses from students who worked with others – P3 – and those who did not – P4 – show that both subpopulations' ratings result in a similar ranking of questions. P4 students saw fewer benefits. Regarding their overall experience, as measured by Q17, P4 students seem split between being neutral and agreeing about whether they would like to use Peer Testing in the future if given the option. The majority of P3 students responded positively. No student opposed being offered the option to use peer testing again.

5.4 Relation to attitude toward testing

Table 7 shows responses from students who used testing – P1 – along with those who only used it when required – P2. Students in subpopulation P2 systematically agreed less to any of the potential benefits. When ranking questions by average rating, both subpopulations kept the same relative levels of agreement.

5.5 Discussion

Table 7 reveals that students' feedback on the benefits of Peer Testing is very positive across all subpopulations.

However, the lowest average agreement rating across all subpopulations is that of question Q16. This suggests that students do not see Peer Testing as better for improving their programming skills than having other students look at their programs and directly point out mistakes. The added requirement for students to resolve their own bugs once they have been pointed out by their peers' tests is the most salient difference between Peer Testing and a method like Pair Programming. There is insufficient data at this point to assess whether the benefits of pair programming regarding individual skill development are equivalent to those achieved by peer testing **[16]**. Further study will be required to determine whether this difference yields a better or worse impact.

Interestingly, students who did not work with others (P4) or didn't use tests when not required (P2) still supported the idea of using Peer Testing again in Q17. This suggests that Peer Testing might have the potential to affect further these students by:

- Allowing those who are not working with others, hence not using traditional group work methods, to still engage with their peers and benefit from it.
- Allowing those who are not using tests to be exposed to the potential benefits of testing via other students' tests.

6. DISCUSSION& FUTURE WORK

Because Peer Testing requires each student to write his or her own tests, it yields the same potential benefits as other testing-focused pedagogies. However, its originality lies in its ties to both collaborative pedagogies, e.g. pair programming **[16]**, and situations where an instructor-provided test harness is available for students to test their solutions **[4][5]**.

The main difference between Peer Testing and having students work together lies in limiting “help” to exchanging tests. This allows students to receive help in identifying bugs while still having to design and troubleshoot their own implementations. While students acknowledged the usefulness of Peer Testing to find bugs, they also communicated a preference for having other students directly point out problems in their programs. From an educator’s perspective, it is essential to go beyond preferences that might be influenced by the fact that some students would systematically opt for the easiest approach even though it might fail to help them in developing more thoroughly their own skills. Therefore, our next step will be to quantify whether Peer Testing leads students to develop stronger individual programming skills than, for instance, pair programming.

Peer Testing is very similar to having students run their programs against the instructor’s reference test harnesses, as done with most automatic grading systems. In both situations, students get additional feedback on how close their programs are to fulfilling requirements. However, Peer Testing provides students with tests that should not be blindly trusted since they have been designed by peers. We believe Peer Testing requires that students devote more thought to understanding tests rather than simply using them as an automatically produced check list. Therefore, while the results of applying tests to a student program doesn’t provide feedback on the programming process itself, it is less likely that a student would be able to arbitrarily modify his or her program until it pass the tests **[8]** when using Peer Testing rather than using instructor tests. Establishing this requires further exploration about how students make use of the results of tests in general, and how they leverage the results of tests provided by an instructor versus other students (peers). Qualitative research designs will be explored to do so.

7. ACKNOWLEDGMENTS

This material is based in part upon work supported by the National Science Foundation under award #0836863 Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] Deitel P., Deitel H.. C How to Program. 7/e. Prentice Hall, 2012. ISBN-10: 0-13-299044-X

[2] Will Marrero and Amber Settle. 2005. Testing first: emphasizing testing in early programming courses. SIGCSE Bull. 37, 3 (June 2005), 4-8. DOI=10.1145/1151954.1067451

[3] Chetan Desai, David Janzen, and Kyle Savage. 2008. A survey of evidence for test-driven development in academia. SIGCSE Bull. 40, 2 (June 2008), 97-101.

[4] Stephen H. Edwards. Using software testing to move students from trial-and-error to reflection-in-action. In Proc. 35th SIGCSE Technical Symp. Computer Science Education, ACM, 2004, pp. 26-30.

[5] Stephen H. Edwards. Improving student performance by evaluating how well students test their own programs. Journal of Educational Resources in Computing, 3(3):1-24, September 2003.

[6] M. H. Goldwasser, "A gimmick to integrate software testing throughout the curriculum," presented at the Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education, ACM, New York, NY, pp. 271-275, 2002

[7] S.H. Edwards, Z. Shams, M. Cogswell, and R.C. Senkbeil. Running students’ software tests against each other’s code: New life for an old "gimmick." In Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE ’12), ACM, New York, NY, 2012, pp. 221-226.

[8] A. Gaspar, S. Langevin, N. Boyer. Constructivist Apprenticeship through Antagonistic Programming Activities, Encyclopedia of Information Science and Technology, 2/e, 2007, Volume 2.

[9] A. Gaspar, S. Langevin. An Experience Report on Improving Constructive Alignment in an Introduction to Programming. Journal of Computing Sciences in Colleges, December 2012, volume 28, issue 2, pp. 132-140

[10] A. Gaspar, S. Langevin. Restoring Coding with Intention in Introductory Programming Courses, Proceedings of the ACM Special Interest Group in IT Education Conference, Oct 18-20, Sandestin, FL, 2007

[11] A. Gaspar, S. Langevin. Active learning in introductory programming courses through student-led “live coding” and test-driven pair programming, EISTA 2007, Education and Information Systems, Technologies and Applications, July 12-15, Orlando, FL

[12] A. Gaspar, A. Ejnioui, N. Boyer. The Role of the C Language in Modern Computing Curricula: Part 1 – survey analysis, The Journal of Computing Sciences in Colleges, Vol. 23 issue 2, pp. 120—127, CCSC Publisher (Consortium for Computing Sciences in Colleges, USA), 2007

[13] Vygotsky, L.S. (1978). Mind and society: The development of higher psychological processes. Cambridge, MA: Harvard University Press.

[14] Tom Wulf. 2005. Constructivist approaches for teaching computer programming. In Proceedings of the 6th conference on Information technology education (SIGITE '05). ACM, New York, NY, USA, 245-248.

[15] Andrew K Lui, Reggie Kwan, Maria Poon, and Yannie H. Y. Cheung. 2004. Saving weak programming students: applying constructivism in a first programming course. SIGCSE Bull. 36, 2 (June 2004), 72-76.

[16] Grant Braught, Tim Wahls, and L. Marlin Eby. 2011. The Case for Pair Programming in the Computer Science Classroom. Trans. Comput. Educ. 11, 1, Article 2 (February 2011), 21 pages.