

# A New Paradigm for Programming Competitions

James F. Bowring  
Department of Computer Science  
College of Charleston  
Charleston, South Carolina, U.S.A.  
BowringJ@cofc.edu

## ABSTRACT

The annual ACM International Collegiate Programming Contest produces a competitive paradigm that is at odds with the pedagogical goals of modern computer science and software engineering degree programs. This paradigm stresses the fast completion of a programming task and evaluates the results solely with black-box testing specified by the judges. In contrast, the pedagogical goals of contemporary college degree programs in computing emphasize the quality of processes inherent in software development and implementation. In 2007, the College of Charleston student chapter of the ACM hosted its annual high school programming competition by turning the conventional programming paradigm on its head to focus on quality-of-process rather than time-to-complete. The judging criteria included both technical and artistic merit. The implementation of the competition emphasized success by giving students working skeleton solution programs. This paper presents the motivation for the new paradigm, the details of its implementation for the 2007 competition, and the details of the new techniques for judging technical and artistic merit.

## Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education—*Computer science education, Curriculum*

## General Terms

Human Factors

## Keywords

ACM Programming Contest, competition paradigm, quality-of-process

## 1. INTRODUCTION

The annual ACM International Collegiate Programming Contest produces a competitive paradigm that is at odds

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCSE'08, March 12–15, 2008, Portland, Oregon, USA.

Copyright 2008 ACM 978-1-59593-947-0/08/0003 ...\$5.00.

with the pedagogical goals of modern computer-science and software-engineering degree programs. This competitive paradigm stresses the fast completion of a programming task and evaluates the results solely with a black-box<sup>1</sup> testing protocol. In contrast, the pedagogical goals of contemporary college degree programs in computing emphasize the quality of processes including those inherent in software development and implementation [2].

This ACM competitive paradigm also informs related programming competitions. For example, every year the student chapter of the ACM at the College of Charleston hosts a programming competition for high school students in conjunction with an annual math competition hosted by the Department of Mathematics. As the new faculty advisor to the student chapter in 2007, I faced the challenge of inspiring other faculty to participate in judging the competition. I soon discovered that there was an undercurrent of dissatisfaction among the faculty: they did not like the competition's emphasis on programs evaluated solely on time-to-submit and a black-box testing regimen specified by the judges. There were also anecdotes from students who had participated in high school programming competitions and who reported that they had been asked by coaches to forget everything they had learned about quality in order to compete.

Inspired by this challenge, I decided to introduce a new paradigm for the high school programming competition that changes both the competitive philosophy and the implementation of the event. The new philosophy emphasizes the quality-of-process over the time-to-complete constraint. Thus, in the new paradigm, the quality of all artifacts is judged both on technical and artistic merits. *Technical quality* refers to objective evaluations of how well the submitted solutions meet the stated requirements. *Artistic quality* refers to subjective evaluations of the organization of the code, the readability of the code and its documentation, and the readability of other artifacts such as output files. Furthermore, the paradigm removes the fastest-to-complete constraint entirely because it is generally antithetical to processes that emphasize software quality.

The new implementation of the competition emphasizes the fun aspects of competition by creating an infrastructure that promotes team success. To this end, each team starts with working skeleton programs for each problem. These programs demonstrate input and output functionality with

<sup>1</sup>*Black-box* testing evaluates only the output of a program in contrast to *white-box* testing that additionally evaluates the coverage of structural criteria within the program.

exemplar files. The teams are to solve each presented problem and introduce their solution into the supplied program. Each problem description also directs the teams to provide test inputs and outputs as well as documentation for their solution.

The main contributions of this paper are:

- Development of a new programming competition paradigm based on process quality;
- Development of a prototype competition based on this paradigm;
- Development of prototype judging rubrics for technical and artistic quality for the new paradigm.

## 2. MOTIVATION

At the College of Charleston Department of Computer Science, faculty discussions have for some time focused on declining enrollments and on how to recruit students into computing. We view our annual high school programming competition, now in its twenty-fifth year, as a key component of our recruiting efforts. We want to attract students who will gravitate to life-long learning, skilled problem-solving, and the pursuit of quality. In contrast to this general goal, we have in the past acquiesced to the conventional model of programming competitions promulgated by the ACM.

Consider, for example, the official 2007 ACM fact sheet that describes a scenario where “[h]uddled around a single computer, competitors race against the clock in a battle of logic, strategy and mental endurance . . . Each incorrect solution submitted is assessed a time penalty. You don’t want to waste your customer’s time when you are dealing with the supreme court of computing [1].” This description’s vision of master problem-solvers working under pressure to produce correct solutions in record time is a flawed model for computer science and software engineering students. This pressure-cooker paradigm may exist in many commercial environments, but commercial practice is not necessarily good practice or a good basis for pedagogy. The ACM itself has recognized that “high-quality [computer science] programs are driven from within the program . . . [while] . . . low-quality programs are driven from without [2].” In other words, it is ill-advised to warp our teaching to fit commercial concerns.

The ACM’s official description of the competition states that “[t]he contest fosters creativity, teamwork, and innovation in building new software programs, and enables students to test their ability to perform under pressure [1].” The current pressure-cooker paradigm for the competition is not the only viable one that this description allows. The historic roots of this current paradigm can be found in the close relationship computer science has with mathematics. The existence of correct and often unique solutions characterizes mathematics and thus informs a competitive paradigm where participants race to find the correct answer. However, in contrast, the software solution for a given problem is not unique. For example, we can solve a single problem by writing different programs in various programming languages. We then compile them with other non-unique programs and subsequently execute them on various operating systems in various operating environments. In fact, two syntactically-identical programs compiled for different operating systems may produce different binary programs. In addition, because the exhaustive testing of a program is

in general intractable [6] we cannot determine whether these two programs are equivalent.

Another manifestation of this difference between mathematical solutions and software solutions is embodied in the notion of the lifecycle of software. The *lifecycle* model of software provides for the evolution of software solutions and thus implies that software solutions are works-in-progress. This concept of continuous evolution and improvement of a program solution leads us to examine and study the underlying processes of program evolution. As computing educators, we teach these processes and emphasize the need for quality in their practice. We teach how to elicit requirements, how to design solutions, how to implement, test, and debug solutions, and finally how to deploy and maintain solutions. This view motivated me to find another competitive paradigm that aligned with the ACM’s official description of the competition. My paradigm turns the current one on its head to focus on quality-of-process rather than time-to-complete.

In the following sections, I describe the competition environment as reified in our 2007 high school competition, and I present the new programming competition paradigm, the details of the new judging metrics developed for this competition, the results of the competition, and finally some conclusions and discussion.

## 3. COMPETING FOR FUN AND SUCCESS

Each year, the College of Charleston ACM high school programming competition occurs on the Friday evening before the MATHMEET<sup>2</sup> of the Department of Mathematics; many students generally participate in both events. The ACM student members treat the competing high-school three-member teams to a banquet and a speaker before the teams assemble in a computer lab for the big event. The actual competition is two and a half hours, thus allowing time for judging before midnight.

For 2007, in addition to the new paradigm described in this paper, we changed the ground rules of the competition to encourage fun and success.

Each of the eleven teams had experience with various languages and we knew from past experience that judging the competition in several languages was problematic. The computer science department in 2007 replaced Java with Python as the first-language-learned and we saw this year’s competition as an opportunity to advertise our choice. Thus, I specified in the announcement of the competition that the only language supported would be Python. To level the field, I provided links to documentation and tutorial resources for the students.

I also implemented two initiatives as a way to encourage success:

1. All problem statements include a working skeleton program in Python that illustrates how to read in an example file and how to output strings to a file.
2. We provide a *Syntax Master* who answers any questions the students have about Python syntax.

These changes were well-received. We also provided plenty of food and drinks to keep the students energized.

<sup>2</sup><http://math.cofc.edu/MATHMEET/>

## 4. COMPETING FOR PROCESS QUALITY

In this section, I present the new paradigm by describing the new process-oriented problem structure and the new judging regimen that measures both technical and artistic quality.

### 4.1 Process-Oriented Problems

Each problem statement is organized to encourage the use of a light-weight and generic evolutionary process model [5]. In an *evolutionary* process the specification, development (design, implementation, and testing), and validation phases occur together and generally quickly yield a prototype that serves as the input for further evolution. Thus, the problem statement in its entirety constitutes the requirements and specification document for the problem. The details section of the problem statement implicitly encourages a design phase, an implementation phase, a testing phase, and a deployment (or submittal) phase. The problem statement begins in the same way that a conventional competition problem begins: with a problem name and a problem description in natural language. For example, the first problem in the 2007 competition begins with two standard items:

**Problem-1:** Triangle Problem (after Meyers [3])

**Description:**

- Design, implement, and test a program that reads lines from an input file where each line is expected to contain three strings separated by at least one space.
- These three strings are to be interpreted as integers that represent the lengths of the sides of a triangle.
- The program outputs a message stating whether the line represents a legal input to the program and, if so, whether the integers form a triangle. If a triangle is formed then the output must also state whether it is scalene, isosceles, or equilateral.

The third part of the problem statement, the *details* is the heart of the new paradigm for the competitors. I present each section of the details below with an accompanying rationale.

**Details-1:**

- The input file must be called “TestInputs.txt”.
- Each line of the input file will contain a test case for the program.
- Additional comment lines may be inserted into the input file.
- The tests should cover possible legal and illegal inputs.
- Technical judging will evaluate how well your tests cover the set of all possible inputs.

This requirement emphasizes that the testing of the program is an important and integral part of the solution process. Furthermore, it hints at the expectations of robustness in the handling of inputs by referring to illegal inputs. Note too, that permission to comment the input file implies that this may be a mark of quality.

**Details-2:**

- The program must be named “TriangleTester.py”.
- The supplied program runs.
- Alter the supplied program to meet these specifications.

- Documentation and readability are important for the artistic judging.
- Technical judging will be based on the correctness of the results when using the input file you supply and on the correctness of the results when using an input file we supply.

This requirement emphasizes the quality of the program itself, beginning with a running program. Since, as described in the previous section, the students are given a running program, they do have a fall-back position. In fact, during the 2007 competition, one of the teams could not solve a problem and instead submitted the skeleton program modified to print an ASCII art heart icon. Testing is referenced again in this requirement and the team made aware that their own testing can predict the judges’ testing.

**Details-3:**

- The output file must be named “TestResults.txt”.
- For each line of the input file, there should be one corresponding line in the output file occurring in the same order as the input file lines.
- The formatting and readability of the output file will be judged for both technical and artistic merit.
- You may include other files at your discretion.

This requirement again emphasizes testing quality by specifying a one-to-one mapping between the input file and the output file and suggesting that the output itself must be a quality artifact. It also suggests that the team may provide additional documentation or files to improve the quality of the submission.

### 4.2 Judging

In this section, I present the rubrics I developed for judging the technical and artistic aspects of the submitted solutions.

For technical quality, there are four main criteria. In the example presented below, these criteria are populated with the actual values used for the “triangle” problem.

**Criterion-T1: Evaluate the submission**

- Does the program run with submitted inputs and no errors? (2 pts)
- Are the required files named as follows: i. TestInputs.txt; ii. TriangleTester.py; iii. TestResults.txt (3 pts)

**Criterion-T2: Evaluate the quality of TestInputs.txt**

- Scalene w/ permutations: 3,4,5; 5,4,3; 4,3,5 (3 pts)
- Isosceles w/ permutations: 3,3,4; 4,3,3; 3,4,3 (3 pts)
- Equilateral : 1,1,1 (1 pt)
- Zero in any position with permutations: 0,1,1; 1,0,1; 1,1,0; 0,0,1; 1,0,0; 0,1,0; 0,0,0 (3 pts)
- Negative in any position with permutations: -1,1,1; 1,-1,1; 1,1,-1; -1,-1,1; 1,-1,-1; -1,1,-1; -1,-1,-1 (3 pts)
- Two sides sum to third side w/ 3 of 6 permutations: 1,2,3; 1,3,2; 3,1,2 (3 pts)
- Two sides sum to less than third w/ 3 of 6 permutations: 1,2,4; 4,2,1; 1,4,2 (3 pts)
- Wrong number of inputs: 1,2; 1,2,3,4; a b; blank line (2 pts)

Team Summary Scores for Two Problems											
Team #	1	2	3	4	5	6	7	8	9	10	11
Problem 1	32	75	69	89	40	78	78	60	67	35	72
Problem 2	0	71	32	28	0	36	29	36	28	44	9
Total	32	146	101	117	40	114	107	96	95	79	81

Figure 1: Scores of eleven teams for two competition problems.

- Non-numeric inputs: a,b,c (2 pts)
- Non-integer numeric inputs: 1.2, 2, 3 (2 pts)

**Criterion-T3: Evaluate quality of TestResults.txt from TestInputs.txt**

- Is there a clear mapping from input to output? (2 pts)
- Are the output results correct? (3 pts)

**Criterion-T4: Evaluate quality of TestResults.txt from JudgeInputs.txt**

- Is there a clear mapping from input to output? (2 pts)
- Are the output results correct? (3 pts)

The maximum technical score is 65. Criterion-T1 is worth 5 points and Criterion-T4 is worth 35 points. I normalized Criterion-T2 and Criterion-T3 by taking their product and dividing by the maximum value of Criterion-T3, or 5 points. This has the effect of devaluing a correct input set when the output is wrong.

For artistic quality, there are also four main criteria. For artistic quality, the evaluations are subjective. In the example presented below, these criteria are populated with the actual values used for the “triangle” problem.

**Criterion-A1: Evaluate the input file TestInputs.txt**

- Readability (1 pts)
- Includes identifying info as comments? (2 pts)
- Includes correct test outputs as comments? (2 pts)

**Criterion-A2: Evaluate the output file TestResults.txt**

- Readability (1 pts)
- Does it include some identifying info as header or footer? (2 pts)
- Does it include identifying info relative to each test case? (2 pts)

**Criterion-A3: Evaluate the program file TriangleTester.py**

- Readability of code itself (organization, variable names, etc.) (10 pts)
- Readability of program with comments (is it well documented?) (10 pts)

**Criterion-A4: Evaluate the quality of the submission**

- Consider whether there is an additional documentation file, for example, and whether this submission would rate an A grade. (5 pts)

The maximum artistic quality score is 35. The technical and artistic scores sum to 100.

## 5. COMPETITION RESULTS

Eleven teams of three members each competed to solve two problems presented with the new paradigm. Each team had the full one hundred fifty minutes to solve the problems. Each team worked until the allotted time was over.

The judging panel was composed of four computer science faculty and two graduates of the department of computer science working in the software industry. Each of the two problems was assigned three judges. The judging criteria were divided into two sets of technical and one set of artistic criteria for each problem. Thus each judge became expert in one aspect of one problem.

The summary scores for both problems are shown in Figure 1. The distribution of the results tends toward a normal distribution, with a mean of 91.6 and a median of 96. The standard deviation is 33.1 and the standard error is 10.0. My conclusion from these data is that the scoring rubrics are well-suited to the job at hand because the scores approach a normal distribution.

This result is for only one competition, and I look forward to applying and improving these rubrics in future competitions.

The students and the coaches as well as the faculty and ACM students approved of the new paradigm and encouraged me to continue in its development.

## 6. RELATED WORK

Sherrell and McCauley also describe an approach to high school programming competitions that emphasizes process over “hacking [4].” The authors created a programming challenge paradigm wherein the contestant teams had one month to design and produce a documented software project in the form of an adventure game. In contrast, my paradigm follows the convention of on-site competition. While my approach shares an emphasis on process with the authors’ approach, theirs does not incorporate any notion of testing. The interweaving of testing processes with development processes is at the heart of my paradigm.

## 7. CONCLUSIONS AND FUTURE WORK

I have presented my new paradigm for student programming competitions and presented my motivation and rationale for the new approach. I have described the 2007 competition as a prototype instantiation of this paradigm. Finally, I have presented my development of rubrics to measure the technical and artistic quality of submitted solutions.

I intend to continue to refine this paradigm and the associated rubrics in future programming competitions. I also will lobby to have the ACM consider this paradigm as an alternate approach for their International Collegiate Programming Contest.

## 8. ACKNOWLEDGMENTS

This work was supported by the Department of Computer Science at the College of Charleston. Tony Leclerc and Chris Starr provided invaluable support and insights during the maturation of this new paradigm. I thank my advisor, Mary Jean Harrold, for her indefatigable efforts to promote software testing.

## 9. REFERENCES

- [1] ACM. Fact sheet - 24 june 2007 - first edition, 2005. <http://icpc.baylor.edu/icpc/About/Factsheet.pdf>.
- [2] Joint Task Force for Computing Curricula 2005. Computing curricula 2005: The overview report, 2005. [http://www.computer.org/portal/cms\\_docs\\_ieeeecs/ieeeecs/education/cc2001/CC2005-March06Final.pdf](http://www.computer.org/portal/cms_docs_ieeeecs/ieeeecs/education/cc2001/CC2005-March06Final.pdf).
- [3] G. J. Meyers. *The Art of Software Testing*. John Wiley and Sons, Inc., New York, 1979.
- [4] L. Sherrell and L. McCauley. A programming competition for high school students emphasizing process. In *MSCCC '04: Proceedings of the 2nd annual conference on Mid-south college computing*, pages 173–182. Mid-South College Computing Conference, 2004.
- [5] I. Sommerville. *Software Engineering*. Pearson Education Limited, London, 2007.
- [6] J. Yang, S. Khurshid, and W. Le. Software assurance by bounded exhaustive testing. *IEEE Trans. Softw. Eng.*, 31(4):328–339, 2005.