NEXT GENERATION COMPUTER SCIENCE ASSESSMENTS

Student Computer Science Initiative | October 2014 to May 2015

What is the SCSI?

The Student Computer Science Initiative leads students in developing a more engaging, rigorous, and growth-oriented CS program by taking community feedback and synthesizing it into proposals, tools, and reports, such as this one. The SCSI was created by Student Council in October of 2014 with support from the student body's most avid CS learners as well as many professionals in industry and education from around the country. More information about the student vision for CS at IMSA can be found on SCSI's blueprint (bit.ly/scsiblueprint).

Why did we choose to focus on assessment techniques?

Of the six recommendations outlined in the Renk and Morrison review of IMSA's CS program, the one on assessments is most striking to IMSA students. Assignments, tests, and final grades have a big impact on the confidence of students, especially those who are exploring CS for the first time. Current CS techniques implemented in schools' curriculum do not allow room for mistakes and judge harshly for imperfections. In order to build the resilience of strong programmers and curiosity of strong scientists, IMSA needs a suite of CS assessment techniques that gives students meaningful and constructive feedback as well as room to explore and revise. Specifically, the Renk and Morrison review calls for IMSA to:

Develop a new and more accurate instrument or methodology for assessing student computer science proficiency

Because no single instrument can fully analyse proficiency, the SCSI has developed multiple instruments that test proficiencies in different, fractional areas. These have been synthesized with insight gained through numerous discussions, workshops, and sample class sessions. This report explains the major lessons that students have synthesized about CS assessment, describes the different types of CS assessments instructors can employ, introduces techniques for these assessments, and includes example assessments that can be used in IMSA classrooms.

What are the characteristics of a good CS assessment?

Strong computer science assessments should:

A. Provide useful feedback, emphasizing **design feedback** over syntactical feedback.
B. Monitor **growth** and encourage challenges at or above a student's skill level.
C. Promote **creative solutions** to problems based on real-world applications.
D. Demonstrate students' understanding of the **problem and solution domains**.
E. Facilitate an engineering design **process** in which students have to make intelligent decisions on the structure of their solutions and the constraints of their problem.
F. Be **resource-blind**, ensuring that access to resources like StackOverflow do not determine whether or not a student can generate a solution to a problem.
G. Build proficiency in **processes** that are essential to programming, including debugging, documentation, **collaboration**, and written/verbal communication of results.

What kinds of CS assessments can fulfill these characteristics?

The SCSI has taken its feedback and created three categories of assessments:

I. **Modeling Assessments**
   - Students use computing to tackle real-world problems, building their capacity to analyse and engineer.
   - Emphasis on characteristics: C, D, E, F

II. **Competitive Assessments**
   - Students, individually or in teams, prepare programs to perform under given conditions and compete against each other, motivating them to take unique approaches to the challenge.
   - Emphasis on characteristics: A, B, C, E

III. **Interpersonal Assessments**
   - Students work with classmates to improve each others' programming abilities by sharing feedback and learning from others' results.
   - Emphasis on characteristics: A, B, F, G

MODELING ASSESSMENTS

Lessons

- Computer science can be used to model real-world problems and solve complex issues
- Computer science can be utilized in all disciplines, not just computer-related ones
- Modeling exercises allow students to verify the correctness of their code with their intuition and knowledge
- Exciting and relevant problems, like modeling the spread of a disease or the motion of a rocket, are more engaging than contrived problems about fake scenarios.
- Special emphasis on assessment characteristics: (C) real-world applications, (D) problem and solution domains, (E) engineering design processes, and (F) resource-blind assessments.

Overview

The power of the computer lies in its ability to do millions of calculations in short amounts of time. For years, scientists have been using computers to generate complex models that cannot be calculated by hand. These models, in turn, have given scientists the ability to do things such as predict natural disasters, determine the effects of global warming, and test solutions to large problems. Modeling assessments give students the chance to apply computer science to real-world problems that they are interested in, increasing engagement and interest in the course. Additionally, students are taught how they might use computer science to solve future problems they encounter in their careers.

Techniques

Level System

Instead of having a single challenge for all students to complete, instructors can opt to base their assessments off of a level-system of increasing difficulty. Each level should build upon the last one as well as offer a new challenge for students. Structuring assessments as such offers challenge to students of all proficiencies and thus improves classroom engagement. A four-tier system is recommended to provide challenge for each student. The first tier should be narrow in focus such as building a single object, which can be built upon in the second tier through developing methods. The third tier should expand the scope of the project and have students incorporate their object into a larger system, and finally, students should attempt to replicate a real-world situation of their challenge in the fourth tier. If needed, teachers may add tiers.

Designing General Purpose Models

In order to make use of their programming skills in other environments, students must learn to develop extensible and reusable code. To encourage general purpose solutions rather than problem-specific ones, students will be asked to model an event or process outside of class. On test day, students will have to use their code to solve a unique problem involving their model. A simple example of this could be asking students to develop a simple algorithm that can encrypt a few sentences of text, and then asking them to use their code to encrypt an entire book, the identity of which is not known before the assessment. A more complicated example might be modeling the progress of a combustion engine based on different kinds of fuel put into it. This assessment values students creating interoperable code that works in a variety of scenarios. Students will be given the scenario ahead of time, but not the specific variables that will determine the expected output. This encourages students to generate general solutions that can be adapted to a variety of problem conditions and expanded on in the future.

Problem Application

Computer simulations are useful because they can be used to predict events in the real world. Modeling Assessments should encourage students to develop models that match the real world as closely as possible. Assessments in this vein could include asking students to use their models to predict the results of a real-world process, then comparing the simulated results to the observed results. This rewards students who create interoperable and realistic models.

Lessons

- Students should learn from others' techniques and implementations and build upon those in order to create better solutions
- Students should understand the full scope of the problem when designing solutions in order to take into account all perspectives
- Collaboration is an essential part to computer science, and proficiency requires a number of skills including documentation, modularity, and careful planning
- Students are motivated to do their best when their competitive nature is engaged
- Special emphasis on assessment characteristics: (A) design feedback, (B) growth, (F) resource-blind assessments, and (G) proficiency in processes are key

Overview

When competition is added to the classroom environment, students have a greater motivation to perform at their highest capacity. However, rather than breeding hostility, competitive assessments aim to provide students with an opportunity to demonstrate where their solutions to given programming problems excel and examine the strengths of the solutions their peers devise. Thinking critically about a problem builds understanding, and competitive assessments require students to think critically both about the way they attack problems and the way their competitors attack problems in order to formulate a superior solution.

Techniques

Programmer vs. Programmer (PvP)

PvP competitions simply place students or teams of students against one another in order to see whose code performs better. PvP competitions work extremely well in assignments that give students a framework to build upon. Students are able to demonstrate their knowledge of coding through customizing their specific competitor within the framework given by the instructor. There are two main aspects of customization that students should be able to control: base traits and behavior. Base traits should allow students to pick starting characteristics for the object generated as well as the standard deviation for future instances of the object, if applicable (like an animal). However, there should always be a focus on creating multiple dimensions of strategy to force student development of more sophisticated algorithms. The more challenging part for programmers to code will be the behavior of the object. The programmers should dictate how the objects should act and what decisions they should make upon receiving input.

Programmer vs. AI (PvAI)

The other option for a competitive assessment is pitting the players against a common AI and testing the proficiency of the project. From there, the results of each student's project can be compared and a ranking can be determined. PvAI eliminates much of the chance that is inherent in having students face off against each other. However, instructors are required to develop this AI as well as make it complex enough to test even the most proficient students' projects.

1v1 (PvP)

In the most basic form of PvP, students face-off against each other in a two-person match. However, this type of competition is vulnerable to the variation in class proficiency. A new programmer placed against a highly-experienced programmer has a high likelihood of doing much worse than if placed against another new programmer. As a result, chance plays an effect on the outcomes of the competition. This can be remedied by placing programmers into a tournament bracket or pool that gives them the opportunity to face multiple other programmers before results are produced (a round-robin or group stage). Alternate tournament structures can be borrowed from professional sports tournaments.

Team-Based (PvP)

Instead of having students individually face-off against each other, instructors can opt for team-based competitions. In the real-world, enterprise development is never a one-person job. Instead, teams of varying sizes and expertises are put together in order to complete projects. As a result, coders must be adept at collaborating with others and using version control software such as GitHub in order to sync changes. Team-based competitions test students' abilities to communicate with one another as well as document their code well enough for their teammates to read. However, the downfall of team-based competitions, like any other team-based assignment, is that certain members of the team can end up putting little to no effort towards their part and bring down the rest of the team. On the other hand, stronger members of the team may end up taking over projects, making individual grading difficult as well. At IMSA especially, the prioritization of individual work in classes over group work in other classes can be detrimental to collaboration in the classroom. To alleviate the problem of unbalanced contributions, teachers need to craft assignments that engage students and require a variety of different perspectives and skillsets.

Arena (PvP)

Instructors can choose to make an arena-style competition, where all programmers' projects are placed into a single simulation to see which one comes out on top. Arena competitions not only get rid of discrepancies that arise from 1v1 competitions, as all students are facing the same opponents, it also forces the students to account for a wider variety of situations as they have to account for the capabilities of multiple other projects instead of just one. However, having many students face off at the same time can lead to random chance highly influencing the outcome of the arena simulation. Instructors should make sure to run multiple rounds of the arena in order to account for this variability, or take steps to reduce the effect of random chance such as using the same-seed random number generator every round.

Player Rumble (PvP)

Instructors can choose to develop a series of increasingly difficult artificial intelligences (AI). The more AI's the student can defeat, the higher the student's grade is. This allows instructors to measure the proficiency of the students as well as set benchmarks for them. For instance, if an instructor plans out his or her curriculum intending for students to get to the eighth level but none of the students do, he or she knows that the students have not grasped the concepts and further class time is needed to go back and review them.

INTERPERSONAL ASSESSMENTS

Lessons

- Qualitative feedback is essential to student growth, but large class sizes make it hard for teachers to provide this to every student on every submitted assignment.
- When CS students are encouraged to share their work and learning with each other, they are exposed to new ways of solving problems and avoid common mistakes.
- Reading a student's code alone is not sufficient to assess their conceptual understanding.
- Making effective pairs or teams for collaborative assignments becomes easier as the semester goes on because the teacher is more aware of students' needs.
- Special emphasis on assessment characteristics: (A) design feedback, (B) growth, (F) resource-blind, and (G) proficiency in processes are key.

Overview

Learning computer science with a class of peers brings many advantages to learning alone. CS classrooms can capture these advantages by promoting the sharing of knowledge between students by providing opportunities for collaborative programming, and even tasking students with helping assess each other. Interpersonal learning experiences strengthen students' learning by exposing them to peers who think differently and pushing them to understand concepts deeply so that they can mentor others. This section presents structures that can encourage, capture, and assess the learning products of these interpersonal CS interactions. This set of techniques makes heavy use of qualitative feedback that is not easy to convert into quantitative metrics. To make up for cases when the qualitative experience and quantitative score may not match up, all of these techniques are designed to be growth experiences in their own rights, so that students can derive personal lessons instead of simply relying on a number for affirmation.

Spotlight Reports

Teachers can lighten their grading load as well as gather useful feedback for their students by outsourcing parts of the assessment process to the class. In a spotlight report, students would be put in pairs and asked to write one page that identifies strong parts of their partner's code as well as areas for improvement. If a peer’s feedback is sufficient, the instructor does not need to spend as much time on their partner’s assignment and can focus on other students’ work. Spotlight reports give students good practice in communicating about code and push them to explore their peers’ work, potentially learning lessons from them. Moreover, spotlight reports press students with the question: What does good code look like? At the start of a semester, the teacher can provide a definition to answer this question and allow the class to modify it throughout the course. The teacher can encourage students to cite examples of the qualities of good code in their partner's’ work, giving reports a set of common standards.

Libraries

Students learning the same topic together are bound to encounter similar problems and resources. CS classes can offer libraries that allow students to share these with each other by allowing them to post and read entries from a common location. For example, a class may have a Bug Ticket Library where students identify frequently-recurring errors from assessments. Not only can the entire class benefit from these warnings, but the students who post Bug Tickets will be especially aware of avoiding those errors in the future. Teachers can monitor which students are active in libraries, encouraging them to both use and contribute so that they become stronger programmers.

Partner Programming

Collaborative projects can be difficult at IMSA because of students’ individual workloads. Thus, partner assignments should be structured so that a student is not fatally undermined by problems with their partner’s code. At the same, the structure of partner projects should promote collaborative discussion about solution design and peer tutoring. Here are some formats that can meet those needs:

- **Code Swap:** Both partners are given the same kind of project, except with different topics. After completing the first level of the assignment, they will switch code with each other and have to program the second level from their partner’s framework. This format reminds students of the value of good documentation. If a student does not finish their first level in time, their partner will work on a sample given by the teacher.
- **Components:** Students are given two basic components that interact with each other and must work on optimizing them. This type of assignment rewards students who design and test their components together, perhaps even learning version control

programs like Git and Subversion. However, if one partner is behind in development, the other still has the original, basic component to test their program with.

- **Cross-Teaching:** Students are given the same problem, but are taught different approaches or content for solving it. After completion, their job is to teach the approach they learned to their partner. The students will tackle the problem again, this time with the approach learned from their partner.

Sans-Code Assessments

Assessing code does not reveal all. Sometimes, students who code with perfect syntax do not have a conceptual understanding of what their program is doing. There are also students who are thinking computationally but struggling to translate those ideas to code. To help students in this cases grow, teachers should employ assessments outside of programming assignments.

One way to assess students sans-code is through interview questions, like those used by companies searching for top-notch computer scientists, not just skilled programmers. Asking students to, for example, explain a useful metaphor for a kind of data structure or write out pseudocode that shows the functionality of a simple game like FizzBuzz, reinforces conceptual understanding and and communication skills.

When students write pseudocode, they fill a blank line in their code with a note about what that part of the program is supposed to do. This is a useful assessment tool outside of interviews as well. The assignments recommended in this report are quite rigorous, so students who run out of time or have ideas for features that they have not yet learned to code should be allowed to submit assignments with pseudocode. Having pseudocode for a feature is preferable to not having it and can help the teacher better understand a student's learning process, helping them push the student to wean off of pseudocode as the semester goes on.

ACKNOWLEDGMENTS

The Student Computer Science Initiative would like to acknowledge the following individuals for contributing their expertise and time to the Student Computer Science Initiative:

Mr. Brian Sea (formerly of Philip Exeter Academy) for providing feedback on the SCSI pilot programs and sharing his experience with IMSA's CS curriculum

Dr. Dong (IMSA) for lending insight on how to blend computer science with its various applications

Dr. Gleason (IMSA) for working with the SCSI and giving valuable curricular recommendations

IMSA Alumni (IMSA) for sharing their vast array[∞] of professional experience & expertise in the real world

IMSA Student Body (IMSA) for providing their valuable input on engaging students and creating great curriculum

Mr. Lawrence (IMSA) for supporting the push to bolster IMSA's CS program

Dr. Morrison (NCSSM) for giving recommendations on the SCSI, providing insight on a Froobles-y future, and sharing his work at NCSSM

Dr. Prince (IMSA) for giving his feedback on the SCSI blueprint

Dr. Renk (North Central College) for supporting IMSA's CS program