The College of Engineering's
Richard and Eleanor Towner Prize for
OUTSTANDING GRADUATE STUDENT INSTRUCTORS
Nomination Materials

INSTRUCTIONS: Please provide the following materials:

1. Cover Page (complete page 1 of this form)
2. GSI Reflective Statement (complete pages 2-5 of this form)
3. Letter of Recommendation from the faculty member who was the GSI's course instructor, including (when possible) supporting quotations from GSI's students

Save them as a single PDF in the order listed. Please use the following naming convention: "GSILastName_GSIFirstName_Towner2017.pdf" (e.g., Jones_Pat_Towner2017.pdf).

DEADLINE: Email your pdf to townerprize@umich.edu by Friday, December 2, 2016 at 12:00 pm.

GSI Information			
GSI's Name:	Pannuto	Patrick	W
	Last	First	Middle Initial
Uniqname:	ppannuto		
Nominating Dept:	CSE		
Course Number:	EECS 398 - Computing for Computer Scientists		
Course Size:	300		
Term:	☑ Fall 2015	☑ Winter 2016	☐ Spring/Summer 2016

Faculty Member Information			
Course Instructor's Name:	Darden	Marcus	M
	Last	First	Middle Initial
Department:	CSE		
Rank:	Lecturer III		

Preparer Information			
Preparer's Name:	Darden	Marcus	M
	Last	First	Middle Initial
Department:	CSE		
Title/Rank:	Lecturer III		

GSI Reflective Statement

1. **Describe a class session, lab, or office hour setting that is a concrete example of your creativity and innovation as an instructor.**

Our class is designed to be highly interactive. We work through semi-scripted exercises as a class together. Occasionally, however, the first running of a script to a live audience does not go quite as planned.

In the middle of the term, we had a lecture on build systems. Tools that understand how to take source code and compile it into a final executable, and importantly understand exactly when and what needs to be rebuilt when things change.

The original lesson plan started with a small project we had worked with a little before in class, worked through its build system rules, explained how they worked, and improved them. Once in class, however, it quickly became clear that working on actual code for an actual project was adding too much complexity and distracting from the point of the exercise. People were getting distracted by the code that was incorrect, rather than the build system. So I threw out the lecture in the middle of lecture.

I invented a new exercise on the fly that got rid of all the coding components and used a build system to stitch together English words into sentences. We were able to see how to define rules for putting components together, explore how to express dependencies to correctly rebuild the sentence (or not), and ultimately achieved the learning goal.

2. Provide an example from your own teaching that gives concrete evidence of your excellence in teaching.

Version control is one of the most important and transformative things to ever happen to software development. It's what allows programs to scale to millions of lines of code and thousands of developers, it's what facilitated much of modern open source software, and it is enormously helpful to even the individual developer on a single project to track progress and eliminate bugs. Unfortunately, it's also one of the most confusing things to use.

I developed a pair of lectures for this course that cover how version control works and start mapping what the underlying tool is doing to the commands you issue. By taking the emphasis away from the more traditional approach of teaching 'what commands do you have to run' and turning it to understanding 'how do you want what is in version control to change', students develop a comprehension about what the tool is actually doing and can develop an intuition for what they need to ask it to do, rather than relying a rote memorization of commands and being at a loss when something goes wrong.

The git (version control system) lectures are by far some of the most popular lectures from the course, and multiple student groups (Michigan Hackers, CSE Scholars) have invited me to come give lectures on git to their members.

The most telling thing for me, however, is that after demystifying this tool and proving that it can be useful, students actually use version control for their own course projects.

3. Describe a concept and/or topic that your students struggled with and what you did to help them overcome these challenges as concrete evidence of your dedication to student success.

One of the things that I have had many conversations about is the puzzle of trying to teach young computer scientists how to debug a problem. Debugging is hard. It's the art of winnowing down from the millions of things that could have gone wrong to exactly what has gone wrong in this case. Debugging is especially challenging when you know less about what is going on however. Too often we have seen students exasperated and overwhelmed when "my code doesn't work" and "I have no idea what's going wrong". It's that last bit that was the reveleation and what I aimed to tackle in desiging a lecture teaching how to debug.

Our class is designed to be highly interactive. I work through commands and problems in real time, I make mistakes, and students follow along. For our debugging lecture, I devised a series of exercises that exhibited different problems that students often encounter. For each problem, we explore what the set of things that we do know are, what we expect to be true when the program starts, what we expect to be true when the problem ends, and how to reason through where our expectations failed.

We started easy, with the type of bug fails cleanly in a consistent place and is easy to track down. But it's when we advance to the harder bugs, the program that seems to run forever but never do anything, that we run into the wall. And this is where I run into the wall with everyone. I say the words, "I feel like I have no idea what this program is doing and what's going wrong" and I let that sit in the room for half a minute. I say the words, "this is really frustrating". And then I start to coax things forward, "well I have to know something...", and I get students to start suggesting the things we do know. We moved forward, and solved the problem.

What was most exciting though, was when a hand shot up a student described a problem that had taken her days to figure out and that she, "[had] no idea how you could possibly debug". It became the perfect live exercise, and together as a class we worked through the same steps.

Ultimately, the most important lesson that's covered by this lecture has little to do with the mechanics of debugging, but the emotional state of debugging: that it is a frustrating process, that is okay to feel stupid, and that you can develop tools and skill set to move forward even in the face of this.

4. Is there anything else you'd like the selection committee to know about your teaching?

To give a little general background, I started teaching as an undergraduate instructional aide here at Michigan. I taught several upper-level courses (482, 373, and 470) and was consistently surprised by the things the our talented junior and senior level students did not know.

While the CS curriculum does an amazing job of teaching the fundamentals and the "hard parts", it had missed a pragramtic bit. Students would type the same 100 character commands over and over. They would lose code copy-pasting to e-mail each other. They would debug by guessing rather than a principled approach. We do an excellent job of teaching theoretical concepts - data structures, operating systems internals, etc -, and the course projects deepen comprehension by compelling students to work through the operation of these things. However, nothing taught the mechanics of moving from theory to practice. We simply asked students to pick things up on their own.

While on one hand, this is "just writing code into a text editor", in real world software development, people don't simply "write code", and for good reason. Computer scientists have developed a panopoly of tools to make their jobs easier. A few students would discover these organically, but most struggle through without them, creating a surprisingly wide achievement gap in project performance that had little to do with student comprehension of the material being taught.

I realized that there was an opportunity here to accelerate the learning and comprehension of EECS students, especially those who came into the program with little or no background or exposure to computer science. I built this class from the ground up and have been overjoyed at the outpouring of positive feedback I've gotten from students. Senior students who "wish they had learned this years ago", sophomores who, "made CS accessible for me" and who had been "thinking about changing majors because it felt too impossible and like I was the only one who didn't get it". This term, students from other departments such as Mechancal Engineering have found and joined the class because, "my CS roommate took this and said it was the most useful thing, and he was absolutely right".

It's my sincerest hope that this course continues to grow, that it can evolve to be a standard component of the curriculum for early career students, and that it can accelerate and improve the learning of every student who comes through the course.

One final element the course stresses is transparency in how it operates. All of the course materials are available online and freely available to anyone, as well as a reference of course concepts authored by current and former students:

W16: https://c4cs.github.io/archive/w16/
F16: https://c4cs.github.io/
Course Development: https://github.com/c4cs/c4cs.github.io