PROFESSOR: Welcome, everybody. So this is the one group meeting that we'll have with all you folks. We really appreciate your willingness to come and help us teach 6-172. Saman will explain what the course is all about, and then I will give a presentation, a little bit, of what your expected responsibilities are, and then Eric, who is one of our TAs, will present what the first project that they've been working on is all about.

We think we have a pretty good acronym, OK, MIT POSSE. Masters in the Practice of Software Systems Engineering, OK, and it is very much like a posse, in that we're trying to sort of deputizing you folks to help us deal with this mass of students that we deal with. There's no way, given the resources that we have available at MIT, that we could possibly give the attention to each of the students examples of code, and so our strategy is to go out and deputize some people to help us teach. As we'll discuss your responsibility is not to do any grading, OK, or even any understanding of the course content, OK? But rather to ask good questions of the students so that they understand what it's like to do software development, and you'll see that most of the students are very immature when it comes to coding practices.

They have taken a minimum of one systems course before this, basically.

SAMAN AMARASINGHE: Probably two classes where they write very small programs--

PROFESSOR: Where they write a very small programs, so they have no sort of sense of writing scale. Moreover the class that we're teaching because it's on the topic of performance engineering, they are inclined to break all the rules in order to get the best performance. And that's part of our game but of course what we want is, we want the countervailing-- you still have to make things maintainable, you really

should do it this way, yeah it's fine to violate some abstractions here and there, but then you should document that you're violating those abstractions and so forth. And that's what we're counting on you for, is to insert some experience and common sense. So, Saman, take it away.

SAMAN AMARASINGHE: OK, good, so to add to what Charles said, when I was an undergrad, when I took something like a science class, or a physics class, or engineering class, most of the problems had kind of right answer and a wrong answer and then it's very easy to grade those things. But when I took something like an English class, all the papers I wrote, somebody really went through carefully and graded it and gave me lot of insightful comments so I can kind of develop my own style of writing. And what happens in software is the programming there's a style that each individual has to develop their style, but they can't develop a very haphazard style so there's a certain set of good styles, certain bad styles. And the problem that is in a large class we can grade for correctness very easily, we can also look at the programs and have a very fast thing, okay, saying, here are no comments, or your comment doesn't seem to be, it's too verbose, or some very high level feedback, but it's very hard to help somebody build their style.

And so what you guys are coming into is to help us get these young students create their own style of programming. The nice thing about this class-- and also sometimes last year the masters found it a little bit hard was, we are giving them small amount of code. They are doing performance engineering on a maximum couple of hundred lines of code. There's no very complicated class structures or something like that.

So we are not looking at the style and software engineering in the macro level, how to structure large classes but more at the micro level. In each method do they have right style? Are they is it clean? Are they commenting them properly, or can they do it better?

So at that level at least the building blocks that they're-- as they go build a larger and larger program they can hopefully use. So that's what we're looking at, and it

helps to instead of dumping you ten thousand lines of code, you only get a couple of hundred lines of code, so you can actually go through it a lot more carefully, and help them out. So this is starting with a crazy eye chart that is our new curriculum that we introduced in the department.

So what you have is some early courses in here, these are basically very general math courses, early courses that do-- they probably write about ten lines of code type thing in these courses, and then we feed into these foundation courses and then after that we sit at, this says 197, actually what should be 172. We sit at that level, so we are sitting at that level, and then the prerequisite for people who are coming is that they're taking a computer architecture course so they know a little bit about architecture. They have taken this introductory software course, so they have written some Java programs, and so they know a little bit about software engineering from that course, and they have taken algorithms course, so they understand algorithms somewhat, so this is the background of the people are coming in.

At the low end there might be some people who have only written couple of thousand lines of code in their lives, and there are other people who are, written half probably-- the Linux kernel. So there are kind of very diversified group of people. But there's some people who it's really new to them, and they don't even understand why you had to do good nice programming, why do you have to document these things. I mean, for them it's a very new concept, because they haven't seen that. So OK, if you're writing ten lines, OK, why do you need document that, I know how to do this.

So part of that is this is, for a lot of them it's eye opening, and in last year also we talked to students, they actually gained a lot from talking to you guys. It's easy for us to tell them do it, a lot of them say OK, just to get the grade, I will do it, but after that I'm never going to do these things, and they just go off in very like, mad at us for asking them to do it, but when you guys say this is what happens in industry, there's a lot more impact on what they're going to do.

And so I would go through very fast, so we have done case study on Matrix Multiply show doing how you can basically do things like algorithmic tricks, architecture tricks, and stuff like that get really good performance on Matrix Multiply. So we started with doing very object oriented, immutable type matrices, Matrix Multiply and then go down to doing blast. How much a performance gap do you think that has?

AUDIENCE: It probably depends on how big the matrices are because--

SAMAN AMARASINGHE: Yeah it is 2k by a 2k matrix so it is, yeah-- So somebody said 200x.

AUDIENCE: 10x

SAMAN AMARASINGHE: Yeah 10x, so actually, in fact we've been-- since I started with the immutable matrix, very object oriented stuff like that, and then did all to optimize I could, I got 300,000x. So this is kind of extreme case, but it illustrates how much, kind of, performance sometimes is hidden in these applications.

And then we talk bunch about, these are done lectures, talk about some very simple things in performance engineering, how to optimize loops, things like how to get the compiler to do certain things. Some basic facts, we have bunch of rules in there that we went through fast. And we are going to do introduce them to performance analysis for a lot of these people. First of all, they don't even know their code is running bad, because they don't know how to look at it other than getting one number then.

Even if they know that their code is running bad, how do you go about figuring out what's going on? They know a little bit about GDB but they don't know anything about performance analysis, so we are introducing them to tools in here. And we are talking about C to Machine Code, lot of them are Java programmers, we already gave them a C primer, saying if you're Java what does C mean, kind of a thing.

And I actually talk about computer architecture today and talk about the entire process they're going to use. They're using this Nehalem processor and all of the

interesting things that happen inside the processor that they had to be aware of. And the same, maybe I will talk about the memory system optimization, and then Charles is going to talk about algorithms and data structures for caches in there and some about storage allocation.

We'll have a bunch of lectures on parallelism that Charles is going to use Cilk Language to illustrate how to write parallel programs, that's what we are going to use in this class. And then more on parallelism, some things like data synchronization issues, dig deep into other performance issues in parallelism in there. And then since their final project is on Ray Tracing, we'll introduce them to a Ray Tracing primer, and see how Ray Tracing is done and so they can go into their final project. Then I'm going to talk a little bit about compile optimization, what compilers can do for you, and how to get compilers to do those things for you.

And a little bit about distributed systems, so how can you scale out on this point? It's about getting a small piece of code to run fast, now how do we get millions of users go to this very large scale system, and get them to work, and what issues are involved. And then we are going to also have about three or four case studies sprinkled around.

We'll get someone from University come and talk specific problems, OK, look I had this problem, and at the beginning it was bad. Here's how I went about figuring out why and here are the things I tried, these are things worked, these didn't work, and how we went go. So kind of a very hands on case studies about doing performance because what these people has to do is develop a methodology. Their own methodology of how to go about solving these problems, and then looking at these case studies will give them a feel how a very experienced person will go about looking at some of those problems. So these are the kind of lectures we are doing in a nutshell.

So the projects we are doing. So students' mission, what we do is we write some inefficient piece of code, not that long, and we give them the specific, here so this is functionally correct but it runs darn slow. And what you can say is, OK, take

advantage of right now, the first project is basically looking at algorithmic and data structure inefficiencies. So OK, trying to find better ways of doing that and change this program to run faster. So that's their thing, so they don't have deal with thousands of lines of code. Small program, it's correct, to run them faster, and this we'll also be asking them to write tests, so you can actually test their implementation, and make sure that implementation are also correct.

And basically there's no right answer. This drives students crazy because they are like, "When am I done?" I'm like, "You're never done until the deadline," because you can keep doing more and more and you could keep increasing performance in there, and of course if you hit very marginal returns and at some point and you might decide, OK, that's good enough.

And also more times the journey is as important as the outcome. So at the end of the day you can say I got it very fast, but how did you get there? What's the process you followed? Some of these programs are easy to get there, but if you haven't learned that process you can't scale it out.

And so, a lot of times your part is to talk to them and say, "What did you do?" And say, "Okay look why did you skip that stage, why didn't you profile it?" And get them to get do that. And some will say, "Yeah, I just look at it, I realize what it is." Like, no just do something there might be something hidden in there, so get through the process, understand the process, that's the important.

So the way the project works is, we give them a project, we start them the project, and there might be, some projects have two parts in there, and final and a project request design document for them to give us, saying what they figure out what's bad and what kind of optimizing they are thinking of doing in this project. And then we have a thing called a beta turn-in. They said OK look we ran this, we got the project and turned the product in, an we call it a beta, because I'll tell you why. And then what we do is, we run everybody's project against everybody, basically we run every project, and we figure out who got the fastest implementation. And we said OK, we announced this and said this is the fastest implementation, and then we

figure out how off are you from the fastest, and we have a proportion of the grade that's basically computer graded. So we figure the fastest, you'll get 100% for something that on the other hand if you are 10x slower you get 10%. Except, the first time we tried, the fastest was 1,000x better than the slowest, so somebody got like 0.1%, that didn't work, so we are doing square root of the former so they'll be little bit more, in the somebody, got 1,000x there.

So sometimes it's because somebody just missed one critical insight and so and also they might not have done the test properly and stuff like this, so we give them opportunity. And then after this beta turn-in you get this code, and the TAs get this code to run and do a very fast check over that and that, and then they are going to have a week to find a time with you, sit down for hour or 90 minutes, and go through this code. And then they will get all the feedback, the performance feedback, and they can talk to people and figure out what they missed and they get another chance to fix all those things, get the performance up to turn in, and there'll be another performance measurement, but now if they match up to that point, they can get a full amount so of course, if people exceed they don't get extra, because we want to limit the competition, but at least you get a chance to kind of match up the best code.

So, what you have is this design review week, that they get to talk to you, and get feedback and, then after that, they fix everything up and then resubmit it.

PROFESSOR: You should mention the beta tests.

SAMAN AMARASINGHE: Yes, so what also we do in this is we're asking them also to create a test suite, and we are going to run everybody's programs against everybody's test suite, and also give them points on coverage of their suite, and also how well their program does against everybody else. So if somebody finds lot of bugs in other programs, or some there's a test that founds couple of bugs that nobody else found we will give a lot of points for that person, because that means they went and did a really good job in creating a test suite. So we are actually using that to basically emphasize on testing and testability issues in there.

And then final, when you're turn in, then the TAs will, at that point, will grade for performance and also the coding style. So that time they will go to the code and assign the grade. And so if the students have listened to you, and carried out what you said, they should get really good grades on that part, and then they had to also provide a write up in that. So this is kind of the flow of the projects we have.

And the project one is a simple algorithmic and data structure trick. So we did things like data rotator, bit flip counter, and this Pentominoes Puzzle that he's going to explain a little bit. So we give these things, that we have a pretty inefficient representation, where most of them has a much better bit representation, and you can do bit tricks to basically get good performance. So these are algorithmic and data structure representations that can have huge performance gains. So this is the first project and this is what next week you're going to get.

So project two is more about memory system performance, things like image rotator and number sorting. We have two parts, the first part is we give them these problems, and they're supposed to run with different data set sizes, look at performance counter results and identify what's going on with these programs. So there's one thing that we'll give them bunch of source compiled, and they have to figure out which sort is which by looking at performance counters and different ones.

So you had to identify, so basically give them idea to basically observe a program behavior, and then we ask them to go and fix some of the performance issues as a second part. So this is a lot about basically getting performance information, understanding a lot of hardware, what's going on and then going and doing some performance fixes. So there'll be two parts, to this project.

And the project three is implementing efficient storage allocator. Most of these students find after they've done the Java and Python, they come thinking there are these objects running in memory, with a thing called pointer that points to, and to understand what a pointer is is sometimes hard. And this is an interesting example that they actually do a lot of pointer manipulations, and you have to get a lot of

those things right and you have to have good discipline to do that, as well as getting some performance, so this is interesting project in there.

Here you don't get too much of performance variance but students have to think very carefully about basically aliasing data structures and stuff like that. So getting some of this dirty little things you have to do, and learning how do it right. And then we go into parallel programming. So first part is introducing the concept of parallelism giving them some simple programs to basically parallelize, and then the second part is we give them insertion detection we're inserting a bunch of lines into a image and then they have to figure out when these lines move when they actually intersect, and so it's not embracing a parallel, you have to actually do by intersegmenting the data into regions and basically figuring out where communication is.

So there's a lot of issue in there to get it done. It's a fairly complex program that they have to to deal with and so they're going to have to think through a lot of parallelism issues, synchronization issues, communication issues, all those things come into play in here. And then project 5 is just a problem set because we want to get to make sure that they understood some basic material we did, so you don't have to deal with that. We give them, actually, a write up they don't have to do any coding in there and they turn that in as a normal problem set. And the project six also you don't have time to critique that because it is delivered the last day of class, but if you're interested in basically giving them some feedback, or, just come in to the finals and see how they do.

So there what they do is, this is a really fun project, we give them a very unoptimized ray tracer, and we okay here's an image that it can generate. If you generate image that looks the same to us, you'll be OK. So you can change the algorithms you are doing, come up with other things you can do, where you can still maintain the visual equivalence and then you can do that, you can optimize, you can parallelize, you can do anything you want and build a ray tracer. So they go and then we explain to them the basics of ray tracing and they do lot of cool things and a lot of these kids actually really change, think through algorithmic issues, even look

at a lot of graphics and find different ways, or better ways of doing things, and come up with really cool tricks and then do really well. So for that we will invite you guys to come to the-- then we will have basically a bake off, everyone will come and figure out who has the fastest ray tracer, and they have the bragging rights, as well as the grade of having the fastest ray tracer, so--

PROFESSOR: I'm going to go through this.

SAMAN AMARASINGHE: Okay, you go through-- we have the schedule here. So the most programs are done in C, and very little C++, not very complex C++ classes, or anything like that. We want them to be very close to the metal. They have been trained in a lot of abstract programming in Java, Python and stuff like that, now they're going to get to the metal, and understand things like pointers, malloc and free, native data types and stuff like that. We're not dealing with things like dealing in garbage collection and bound checks, and all the other things, so they're really seeing what's going close to the metal, and sometimes we are even encouraging to look at at a little bit of assembly work, that's not really required.

AUDIENCE: Have these students had exposure to any kind of assembly language programing?

SAMAN AMARASINGHE: In 004, they are using some assembly programming for in the architecture class, but I think it's a very simple, I think RISC machine inspection they're doing. And then we will have our course website as you go about you can look at what the course is doing and also if you change this F10 to a F09 you will see last year's code. So you can actually even see into the future what we have been doing. So if you want to kind of preface from the front.

AUDIENCE: I took a look at this website and found that most of it was unavailable because of--

SAMAN AMARASINGHE: We should have Word readable we need to make it if it's not Word readable--

AUDIENCE: It says it's Word readable. To us.

GUEST SPEAKER: Some of it is a lot of it was not I've said you need to, I mean it's all available for-

AUDIENCE: Certificate?

SAMAN AMARASINGHE: We will check that, because all this things that we have set up in the background to get everything Word-- we want to make everything Word readable, so we'll make sure that happens. So, OK, so that's all I have about the class and one thing interesting more so this projects, the best to worst can sometimes be 1,000x difference. So what that means, even for the people who are struggling, they can get something out of the class.

But unlike other classes where you just read the limit and then you're bored, you can really keep pushing the limit, especially things like the final product, the people who did well really pushed the limits, they could have done that for a graphics class type thing. I mean they pushed algorithms changes, did amazing parallelization techniques, data partitioning, everything. And where some people just kind of stayed within that and did some small parallelization got some like low hanging fruit in there.

I mean what we're trying to do is, computer science these days, we get some students, as I said, their first time they're coding anything more than ten lines and other people who are already a submitter for open source projects, who have written millions of lines of code, we have these two. And these codes, I think we are trying to cater to at every level, and keep everybody basically very motivated, and excited to be in there. And I think the last year it was a pretty good success. And they really enjoy meeting masters last year, and after we go to discussion we can talk about that a little bit.

PROFESSOR: So we have two websites, as Saman mentioned before, so here they are, and then here's the schedule. So we'll run through this, in more detail for what is expected of you, in a moment, but basically there's four projects that's sort of are organized along here. Oops, that typo didn't get fixed. That's should be November 19 obviously there, instead of October 19 if you want to correct that on your calendars.

OK, so in general it begins with an information session which you're allowed to attend in person but most people just dial in, it's more convenient. Then we will

provide you with the beta code submitted by your four students and then you'll have two meetings that typically are 60 to 90 minutes with each of your two pairs. OK, so let's go through how this is working. So the first thing is just to make sure you folks recognize that you're making a commitment. It's very difficult for us, and it's even more difficult for the students, to cope with something that happens mid-term, OK?

Where suddenly their master disappeared, or whatever, so we're very big on anti-flakiness, if you will. And so if you believe that there's going to be a problem please let us know as much in advance as you can. Sometimes things are unavoidable, they come up, we don't fault you for that or anything, OK? Just let us know, we'll try to deal with the situation. But for the most part we're really asking you to make a strong commitment to the class, and so if your situation is such that you're unsure it's best to talk with us or politely decline to participate, rather than putting on the rosy glasses, say, "Oh yeah I think I can do this, even though I know that I've got a major deadline that conflicts with one of these projects," or whatever.

It's just really difficult on the students when things change. They are sort of going I don't know if you remember what it's like being in school as an undergraduate, but pretty much the students go sort of hand to mouth, in terms of like, they look and they say, OK I got this deadline this week I better work on that one. They don't really look ahead and so forth, and so it's up to us, to some extent, to recognize that they're in that mode and that we need to do things to make that easy for them. When they get sick or something mid-term it can really throw a student way off for the term, and so certainly when something happens to the staff, that sort of, at that caliber, can really knock them off their trajectory.

The design and code reviews should normally be held at MIT, and the students will have some locations. If you are within walking distance to campus however, you may suggest an alternative place to meet, such as at your own workplace, if that's more convenient for you, OK? But if a student balks at doing that you shouldn't pressure them to accept. The content of the off campus meetings should be professional, it's strictly the review process, no lab tours, no free lunches, or dinners, or what have you, OK? And we'll talk a little bit more about that later.

But the main thing about them is that they can be scheduled in the evening or you know whatever time you and your two students find convenient to get together, OK? There's no limits as to when you know what time those can be. They can be on the weekend they can be, whatever you can work out, very flexible.

So here's the basic guidelines, each master is responsible for reviewing the code produced by four students. The students typically will work in teams of two or three for the projects, OK? So what you will have is your four students will be organized into two review groups of two students each, but none of the students will be on the same project.

The reason for that is that we want to make sure that every student who goes to represent their project can represent the whole project. So they're responsible for the whole project. They don't have a buddy there on the same team, who can cover for them when you're asking them questions about their project. You ask them questions they've got to be able to answer about the whole thing, even if they didn't code every piece of that. We're encouraging them generally to do things like pair programming, so they really should understand what's going on throughout.

The reason for having two groups, rather than meeting them individually, it turns out there's actually some good knowledge of one of them sort of being able to see what the other teams are doing. So generally they're not allowed to share their solutions, et cetera, with other teams, but in this meeting that's OK. The ideas and so forth can come but they shouldn't be making copies for each other of the code of the other group. But they can say, oh is that what the other team did, And so forth, and there can be some learning for them to submit the final project.

SAMAN AMARASINGHE: Also last year, lot of masters very successfully used the other team members to kind critique each other. So showing something do you understand this, and then they can look at it and say, "Yeah, I can see what's going on, that's very useful," also when we go to masters we can talk about of that type of thing.

PROFESSOR: Also things like documentation. You can look at the code there and ask the other

one, "Do you understand what that documentation says?" And so forth. It turns out it's actually easier than that, because usually if the documentation is inadequate, the student himself has trouble remembering what they did, right? I mean you know how this works with code, so.

So consequently you're responsible for generally for two review meetings for each of the four projects. Now as it turns out we'll assign things, we're going to have things like students dropping the class, and so there will be some minor reshuffling as we go on. Some of you may end up with only one team of two students.

Some of you may find that you're reconstituting it. In some cases if we end up-- we may ask somebody, "Can you cover an extra team this time because somebody is going to be out of town?" Or something like that. So there may be some variations on this, but this is basically how things go.

Now, one thing you should understand is that you are not going to grade the students. So, one of the things I learned long ago was that when you grade somebody they behave differently than if you don't. So you're not grading them that has a plus and a minus.

The plus is that then this is really pure feedback for them. This is to help them, they know that you're supposed to have read it. The downside is some students will say, "Eh, then I don't need to listen, because nobody's holding a stick over my head." But the reason we do that is twofold. One is because we think that the quality of interaction is better, and the second reason is because you're not empowered by MIT to give grades. So you won't be grading them.

You're also not responsible for the technical content of the class. We're going to be teaching stuff, hopefully which is sufficiently new, some of it, that some of you may not know all of what we're teaching with respect to parallelism, and so forth. Some of you may that's great, but that's not the role. We will take responsibility for teaching the content of the class. What we're really after here is for you to listen as an experienced program developer, and provide feedback to the students about software engineering.

In other words explained to them what's going on. "Why didn't you write that in your documentation? How come all your variables have only one letter? Why is this stuff wrapping around three lines before you-- why no white space?" This kind of stuff.

So, remember they're very smart students, but they're also very immature many of them. Some of them you'll discover are like way mature, but there's going to be some of them that are just absolutely this is new to them, and what they need is some encouragement and some suggestions as to what they can do to improve the quality of their code. And since we're not in a position to review it, why we're deputizing you as members of the MIT POSSE, is to deputize you to help them produce better quality code, which is not something that's easy for us either to give feedback on, or to grade.

So hopefully when we are grading them in the end on their final submission and the quality of things, it reflects that they've learned something from you folks about producing good code. Also you want to be careful, this is one of the bugs I have when I interact with the students is, I tend to do most of the talking if I don't check myself. And one of the really good things is to have a guideline that they I don't know how many of you have teenage kids, and if you lecture them how quickly they learn right? No, they don't.

So the goal here is to help them learn. And so lecturing usually doesn't work. What works generally is getting them to do the talking.

And then it's great to pepper your interactions with anecdotes. I work with one person who did this, or early in my career I did this and here's how it blew up in my face, that kind of thing. Excellent to have anecdotes that you can give with real world experience.

Why you might think it's like this, but in fact, if you do it that way you're headed down a very bad path. Let me tell you story about that. That's fine, OK.

SAMAN AMARASINGHE: Or if you find a very stubborn person saying, "If you do that I won't hire you."

PROFESSOR: That counts too. You would never get a job here if you coded like that. And in fact that's actually what Saman says, that's actually one of the big benefits of having folks from industry providing that feedback, because we can say that, if you do it like that you wouldn't be hired by XYZ corporation. You can say that, but if you say, I wouldn't hire you, if you did that, that has a, it's like, oh! It has a big impact, whereas when we do it it's like eh, you know. It's like what do they know?

So the guideline is make them do a lot of the talking, so that they're trying to explain and you teach them a little bit what a design and code review is. Now, in fact what your review is, is both design and code review.

SAMAN AMARASINGHE: To add to that, what we have told them is to prepare for what a filing statement type thing, to walk through and explain what they did to you guys. So they will come with some preparations stuff that they can do that, let them, both of them kind of do that, and then start them off the instructive part.

PROFESSOR: And maybe they'll do that. But that's the goal is, you should say, "Look, when you come you should prepare something that brings me up to speed," and teach them a little bit about what's expected of them when you have a design or code review in industry. So for each of the projects, as we mentioned, there will be a short information session, typically by dialing conferencing, except for this one, which we'll do in a minute.

After the students have submitted their beta releases we will get the code to you. Last year we had the students responsible for getting the code to the MIT POSSE, and that was basically a nightmare for everybody involved. So this time we will make sure that we get you their code, and I hope that way the unreliability-- I don't know if you remember but like, these are mostly like juniors and seniors, and when you're juniors and seniors you're not 100% reliable in everything you do and your word is your bond is not necessarily, oh I forgot that my word is my bond, kind of thing, right?

So by us getting it to you, we'll remove one layer hopefully of flakiness. Next what

you'll do is, you'll coordinate to meet with a mutually agreeable time, and then after the review you're going to provide us with a little bit of feed back, it doesn't have to be long, just a short email, not as a grade or evaluation, but to help us understand where the student's are doing well, and what they're missing. So what we may need to stress in class and so forth.

And then, of course, after the review, they have the opportunity to make changes to their code before they submit their final revision. Now, that's a little bit different from industry where you would go through typically some kind of feature freeze, and code freeze, and so forth. Our beta is beta sort of in name, but not necessarily in all aspects, in that they can actually completely rewrite their code between the beta and the final version.

SAMAN AMARASINGHE: So in your feedback you can also tell a little bit about in your interactions what worked and what didn't. This is the first time, as far as we know anybody's doing this kind of a thing anywhere. We are also developing a process so good practice is very--

PROFESSOR: And since your email will go to MIT POSSE and everybody will read it, what hopefully you can do is, some of the practices that you find useful, I tried this at work, I tried this, it didn't work. That starts to get spread out among you and is becomes very helpful for each other to say, "Oh, well that was an interesting idea let me try that." And so we hopefully we'll be able then to start compiling some best practices.

AUDIENCE: Will we be identifying students by name in those broadcasting minutes?

PROFESSOR: Yes, yes, in fact here's the thing is, what you're going to do is give, well it one, names the students who attended, and that helps us keep track of who's actually going to these and who isn't. The strengths and weaknesses in the student presentations. Some suggestions for improving the assignment. That's suggestions for us to improve the assignment for them, and any other comments.

So we got some really good feedback last year that, "Hey the code you're handing

out as the initial code that was really sloppily documented." Last year we were much more behind the eight-ball in terms of trying to do this. This time we've had more time, and in particular we also have some really outstanding TAs who've worked very hard on the software engineering of the project. So I think that we'll be better in that regard, this year, than we were last year.

AUDIENCE: I just had a question

PROFESSOR: Yeah, sure.

AUDIENCE: You mentioned that they're going to give to you and you're going to pass the code out.

PROFESSOR: Yes.

AUDIENCE: Do you mean just the code or are you thinking structured more as a design review, they really will be talking about, a review package. It would be the code plus I don't know whether you're doing documentation generation or any sort of an intro or front matter?

PROFESSOR: Yeah, so the question is whether in addition to the code there should there's going to be any other documentation, or package, or whatever. In fact, most of the final documentation is going to be for the final project and not for the beta, and so they should have things well documented within the code, for example about what their strategies are, and how they work, and so forth, but there won't generally be a write up. They should be providing you at your meeting with a brief outline of what's going on, and you can ask them after the first meeting for any other thing that you think would be helpful to them and to you in making the design meeting more productive.

One of the things always in these things is how much do you require out of the students and at which time. They have in this project, we're on a relentless series of deadlines for them. And so we've tried to do as much as we can while being still relatively minimalistic in asking for too much stuff too early. And so it would be great if they had a complete package of here's what's going on, the fact is that they were up really late the night before trying to get this stuff done, they've also had to submit

beta tests and so we decided that the final write up of how their project worked, et cetera, we would save for the final, and it won't be something, in English written like that will only be in the code, it won't be as a separate package.

AUDIENCE: Right.

PROFESSOR: Any other questions about that? OK, so one final word, relationships with students. So one of the things that MIT has, and many other places have, I'm sure that you're aware, that if you're in a power position with somebody, and this in particular applies at MIT with students, you must ensure that all your relationships are strictly educational.

And what that means is like a, no romance, OK, duh. But it also means in this context, no recruiting, hints at job opportunities, offers of summer internships, lab tours, et cetera, free dinners, and so forth. Your job has to be completely educational.

However when the term is over you're welcome to continue your friendship in a non-power relationship. There's no longer a power relationship you're welcome to start a romance with one of the students if you will, but in particular, allowed to seduce them to come to your company as a summer intern, or a job or what have you. So you're welcome to do that but not during the term, those sorts of things are absolutely off bounds.

Now the students themselves may say, "Hey do you guys have summer internships, and so forth?" And it's up to you folks to draw the line and say something like, "Yes, we do, but if you're interested in that, we can talk about that after the term is over," for example. So, put them off in some way, and return the conversation to the educational mission for the term. Yeah?

AUDIENCE: So what if the students do a good job?

PROFESSOR: Yes, but then you run the risk of it being interpreted as a quid pro quo for employment, which is exactly the thing that we're trying to steer away from. They're smart MIT students, I think they'll figure out that if they're interested in a summer

internship they should be impressing their mentor. Not all of them figure this out, by the way.

AUDIENCE: So just to clarify. Just to clarify, it's okay to say, "If you write your code this way, I won't hire you." But it's not OK to say, "I wouldn't hire you." I just want to clarify.

PROFESSOR: You'd say I wouldn't hire you, OK, my company would not hire somebody who wrote their code like this. You could say that.

AUDIENCE: But it's not okay to say my company looks for people who would write code this way?

PROFESSOR: No, I think that would be OK too. No, the issue is whether you personalize it, right? If you're interested in a summer job, I can get you a summer job, all you have to do is, dadadadadadadaaa, and now they feel like there's another master to serve. So what we want to avoid is having them feel like there's another obligation on top of the obligation they have to the class. They should not feel that in some ways they're obliged to do something for you in order, for example, to get a job.

SAMAN AMARASINGHE: Or get a good grade.

PROFESSOR: Or get a good grade, or--

SAMAN AMARASINGHE: If I say no to the summer job they're going to badmouth me and I will fail this class or get a bad grade, and then. So, that put them in a binding situation that they won't work for the class, but they'll feel obliged to something else.

PROFESSOR: You're in a power situation, and you're in a situation where you must not let them have any sense that there's any power relationship there, if you will, in other words--

AUDIENCE: We shouldn't try to extort meals from them.

PROFESSOR: For example. Exactly, exactly. Also if you feel anything awkward is developing, let Saman, or me, this is something for which the staff only email is better than probably the MIT POSSE one.

SAMAN AMARASINGHE: Or even to us directly.

PROFESSOR: Or even to us directly, although I actually think it better to go via the staff only. It really is much better to go get via staff only, because the TAs need to know what's going on with all the students in class as well. Question back here.

AUDIENCE: Two logistical questions but I don't want to interrupt you before you finish.

PROFESSOR: No, that's fine.

AUDIENCE: Okay, it's not clear to me do we meet with the same four students throughout the term?

PROFESSOR: If possible we will do that. Because otherwise what happens is, they hear it from one and then they go to another. So this way there's some amount of continuity, but that won't happen perfectly because students will drop and we'll have to reconstitute groups and things.

GUEST SPEAKER: The winners get David and the losers get me. That's OK. I just wanted to be clear. Also you e-mail the code to us before we do the sit down session?

PROFESSOR: Yes.

GUEST SPEAKER: Are we expected to review it by ourselves beforehand?

PROFESSOR: I think it helps to take a look at it, yes.

GUEST SPEAKER: As opposed to walk in cold and doing just that.

PROFESSOR: Yeah, yes, that's right, I mean I think you do a better job if you have looked at the code beforehand to see, oh my goodness this is, and take a few notes. But for this we're relying on you to use your best judgment for what's the best way of giving feedback to the students. But I think generally the reason we email it to you in advance is specifically so you have a chance to look it over.

Once again we're not so concerned about whether the content of what they're learning in it as much as-- you look at it, it will take you two seconds as soon as you see one piece of undergraduate code, it'll take you two seconds to say, I don't care what this thing does this student needs to learn X. I mean you will see. Is that the feedback, from people who took this--

AUDIENCE: You'll know it when you see it.

PROFESSOR: Any other comments from people from last year what--

AUDIENCE: Yeah, a couple comments. One very minor sort of procedural things. Personally, I found it a lot more useful to me to show up in person for the information sessions than do it by conference call, if for no other reason then a lot of people who called in had issues with voice quality. You know, just getting a good quality voice signal over the phone.

Also I found it very helpful to have each student review the other one's code. So I would just do that to start with, just hand it to each other and here, what do you think? That also gives them some experience of actually doing a review, as well as being the person who's work is being reviewed.

PROFESSOR: That's a great technique just because one of the things that you want them to do is empathize with the person who's going to read the code, and so, by putting them in the situation where they have to read somebody else's code they very quickly learn oh, this is, my goodness that's exactly what I did to the other one, so that is a great technique.

AUDIENCE: The reference implementations are we going to get those--

PROFESSOR: Yes.

AUDIENCE: Soon? Early?

PROFESSOR: Yes, those generally we will be able to give you the reference implementations at the information session. Or actually, we should probably get it to them before the information session.

AUDIENCE: I would suggest before the--

PROFESSOR: Yeah, let's get those make sure we get those to you before the information session. We'll get you a reference implementation at least a few days in advance.

AUDIENCE: OK, also something that was a recurring issue last year regarding the reference supplementations which was the... there was some issues with some of the reference implementation--

[INTERPOSING VOICES]

SAMAN AMARASINGHE: --software engineering.

AUDIENCE: In particular things like functions that should have been local to a file they were not declared static or, that kind of thing, or macros that should have been used or that were used and shouldn't have been.

PROFESSOR: Last year we were running with a much larger class than we expected, with no infrastructure to speak of, and we had two TAs who were flat out, and so we definitely cut corners. This year the department has been good to us, we have four TAs, and so I think that the quality is going to be higher. That doesn't mean that the TAs are seasoned 20 years of software development experience or more, either. So they have some things to learn too, but already based on the first project we're ahead of the game.

AUDIENCE: OK, also if from time to time one of us wants to attend the class?

PROFESSOR: Feel free to attend the class.

AUDIENCE: OK, where's the schedule?

PROFESSOR: It is Tuesdays and Thursdays in 26-100 at--

AUDIENCE: Is it on the website?

PROFESSOR: It's on the website at 2:30 to 4:00. So Tuesdays and Thursdays 2:30 to 4:00, you're welcome to come and learn all about performance engineering. Reed, did you have a question?

AUDIENCE: No, I guess I was just going to say that in regard to the feedback, we TAs here have got that.

SAMAN AMARASINGHE: During class last week most of the TAs were students last year, so they aren't learning anything new.

AUDIENCE: I'm sorry, where is the class?

PROFESSOR: It's in 26-100

AUDIENCE: So it's not here?

PROFESSOR: No. So this is the situation, we outgrew our classroom which seated 60, and we had 85 or 90 students, so they put us into the next smaller classroom that we would fit into. That was 26-100 which fits 500. Then we found that in 26-100 you needed a microphone for the students to hear you, even though we had them all crowded into one quarter of the auditorium, you still then-- just the structure of it-- you needed a microphone, so they said, "That will be $85 a lecture." And you say, "What do you mean $85 a lecture?"

SAMAN AMARASINGHE: Someone has to walk in just to turn on--

PROFESSOR: To turn on thing thing. So, anyway we sorted all this out OK, but it was sort of funny that you're charging us for being in a room that we have no choice as to be in? Anyway it was very funny.

AUDIENCE: Are your lectures video taped are they available--

PROFESSOR: They are videotaped. We are not planning to put them up quick because what that does is discourages students from attending the lectures, as you can imagine.

SAMAN AMARASINGHE: Mainly for OCW OpenCourseWare So they basically they could go to OpenCourseWare they will process it and they have to go through a lot of legal stuff and make sure the named are right.

PROFESSOR: Are right and so forth, yeah. There is a bunch of stuff there. Good, so it says final words here but they aren't really final, because we still have Eric to talk about the first project. But the feedback from the students last year was that the MIT POSSE was one thing that made this one of their favorite classes. It was getting that kind of seasoned feedback in a very small group setting when you look at the student faculty ratio, that's one thing you're always battling. especially at MIT, where we are in a very popular major, and so the ability to recruit all you folks to help that ratio in favor of the students is really very much appreciated by students and by us, so enjoy the term. This is really rewarding the people who have done it before we had just tremendously positive feedback from them about just how rewarding it was to work with these young programmers.

SAMAN AMARASINGHE: They are very smart people, you can see the changes, you will see the you impact on them. You will see you're molding these people into better programmers, so that's the fun part.

PROFESSOR: So Eric is now going to talk a little bit about project one, and this is similar to what we will do, although most of you will probably be on the phone unless you heed Barry's advice, for the future one. So I'll get out of your way.

ERIC: So the project one this is, I think, the one project that was substantially different from last year. So, the code for this project we, the TAs, coded up this semester. Did we give out the hand outs, Saman, or Charles? Yeah so these are the--

PROFESSOR: So this is the actual assignment and we'll share this with you, plus the reference implementation, I think.

ERIC: Well we'll send out the reference implementation tonight then. In fact, now people have already done their betas for project one. Project one has two parts, it's a part called Every Bit Counts and a part called Tiling a Torus with Pentominoes.

So the first part it's an abstract data types of modeling and bit array, packed bit array and they are a given a couple of tasks to do on long strings of bits. So you might have eight million bits and there are two functions to rotate the substring of the bits by some amount to the left or right, and the other operation is to count how many flipped bits there are in a given substring. So the specific interface for this, this an excerpt without the comments from the header file of this C module. There is a couple of accessors which accesses bits individually those are not, well those accessors are currently used to implement, in the reference implementation, the rotate and count flips operations.

So what they will need to do is to go into the C module, the implementation, and change these functions, to basically be 10,000 times faster, or whatever, I can't remember what they actually got. In particular the rotate operation is incredibly slow, there's one constraint for them and we're saying that, well imagine that this is on the cell phone or something, where you have a really big bit array, but you don't really have very much more memory to consume beyond that big bit array. So we're asking them to only do this using a constant amount of memory.

So that's the first problem with these two parts, the rotate and the count flips. Any problem, questions about that particular part? Does it seem clear what?

AUDIENCE: What does the count flips do?

ERIC: Oh yeah, so I should've swap those two slides. So, if you have a bit string, suppose nine bits or something, 1, 2, 3, 4, 5, 6, 7, 8, that would have to be 8. If this is the substring you want to count, then you count transitions from zero to one, or one to zero. And there's some new tricks you can do by for instance, considering six--

AUDIENCE: Count the number of runs, of consecutive value.

ERIC: Yeah, the number of runs.

AUDIENCE: Plus plus one-- or minus one, yeah.

ERIC: That's right. And you don't count the beginning or the end as a transition, so in this

particular substring there is one there, one there, and one there. And it's not a substrings, it's always substrings, it's not the entire string, and you can see that for instance, you give us an argument, the bit array and you say where the starting offset and the number of bits that the substring that the substring is and that's where the operation is done. So our testing routines will do lots of rotations in different parts of that substring.

AUDIENCE: For the memory constraint, do you have a fixed size that they're allowed to use or?

ERIC: We decided against the fixed size. We just say don't use malloc, and you can allocate constant amounts of memory, and the thing is if they choose to have a 16 megabyte or four gigabyte lookup table then their performance will suffer anyway. So yep?

AUDIENCE: What is the computer architecture?

ERIC: What's that?

AUDIENCE: What is the computer architecture?

SAMAN AMARASINGHE: We are running on-- we got this donation from both Intel and Dell, we have 16 machine-segments, each with two six core processors and 48 gigs of memory. So they're doing it on 12 cores so it becomes interesting, and you got a parallel 740 gigs, so we have a lot of memory in these kinds of things and we'll be asking them to run through that. So these were the latest what's the word?

PROFESSOR: Core i7.

SAMAN AMARASINGHE: Core i7s.

PROFESSOR: Nehalem.

ERIC: For this first project, we're asking them specifically not to parallelize.

AUDIENCE: It's all on Windows or Linux?

ERIC: Linux.

AUDIENCE: Linux?

ERIC: Yeah. So the idea here is to exploit bit operation and in particular in rotates. The reference implementation does the really stupid thing of you want sort of 4,000 bits to the right rotates, well I'll do it one at a time, that's the reference implementation. But they're only allowed to us a constant amount of memory, so there is certain amount of tricks you have to do to do it in a more efficient way.

AUDIENCE: Do you stress in the course about portable data code from one operating system to another, one machine to the other.

PROFESSOR: This is where we want you to stress the importance of it, because that's not something that we're in a position to do or enforce, or what have you, OK? We let them assume for this problem, for example, that the machine they're working on is Little Endian. We talked about the issue, but we did not insist that they wrote code that would be that would port to a Big Endian machine, because they're running on the Intel machines.

But this is exactly the kind of thing where code quality would say, portability, maintainability, readability, all these kinds of things, this is exactly the kind of thing we want you to say. You say "Hey, you did it like this, in fact did you know that without sacrificing anything in performance, you could have made it portable from one operating system to another?" And then you could talk about those issues, and that's absolutely super, to do that. But we are not spending a lot of class time doing that, we're really focusing on the performance engineering part of the course.

ERIC: The reference implementation should be portable, so this should be portable, although we did make sure here, for instance, in the reference representation of bit array here, that the order of the bits within each bite happened to coincide with the low end ideas of these machines. So that if you happen to cast it to a 64-bit integer on each of the bytes in the array then you would get something that was easy to work with on a low end in architecture. But the reference implementation is portable

itself, or should be, so tell us any kind of problems with the reference implementation like that.

The next problem is based on the puzzle, it's a puzzle solver. It's this puzzle that I'd never heard of before. Apparently, if you generalize dominoes you get Pentominoes and the five end cases. So there are this many configurations of dominoes with five squares on them, I guess. And the pentominoes puzzle is you're given an eight by eight board with four squares crossed out, and now you're supposed to arrange these so that the entire board is filled up, and only-- what's that?

SAMAN AMARASINGHE: You can do flips of these--

ERIC: Oh, right. You can do flips, you rotate

SAMAN AMARASINGHE: And they will dissipate and all those things.

ERIC: Right so the task is find all the solutions, although I believe in our reference implementation that was too fast, so we decided to generalize the puzzle to the toroidal case, where the board wraps around the edges, just to make it take a little longer, to do.

PROFESSOR: Like space boards right? You go off the right edge you come in the left edge.

AUDIENCE: Top and bottom also?

PROFESSOR: Top and bottom also, yeah.

ERIC: So these two are both examples of solutions to this puzzle.

AUDIENCE: I've got a question here.

ERIC: Yep.

AUDIENCE: Are you allowed to put the empty space where ever you want?

ERIC: Well, that's the input to the solver.

SAMAN AMARASINGHE: Given these empty spaces--

AUDIENCE: OK, and then it fills in the rest.

ERIC: That's right. Of course you can put empty spaces so that it can't solved. And then they have to correctly determine that there's no answer.

PROFESSOR: That there's no answer.

AUDIENCE: Right.

ERIC: And, also I believe the reference solution takes days, for certain. To find all the solutions of certain configurations of the initial four dots. So there's an option to the program we give them, to just find the first 2,000 or something.

PROFESSOR: So once again you can see this is conducive to bit tricks, where you represent the board as a 64-bit word and then do masking operations, and so forth, to determine whether a piece fits and do clever shifting. It's easy to shift a piece down one row, where you're trying it out, because that's just a shift by eight, but how do you shift it horizontally, to come around on the other-- So they have to do some shifting, some masking-- So anyway, the first project is all about thinking about bit representations of things, and so forth.

ERIC: In fact, we give them a solver that we don't really-- well, a searcher, routine to search the space of possible solutions, that we don't really expect them to change, but they can change it if they want to do. So, the interface here is basically a board data structure that they're free to change the internal representations of.

Or did we somehow prohibit that due to testing? No? No, no, no, that's the whole point, right. So again, you have this basic accessor for use by the testing routines. But then what you really need to, well, you need to change basically the entire implementation of all of this, if you're changing the representation of the board, for instance.

But this is the function that should be running faster after you're done, so I guess that comment is misleading. This is the function that is supposed to run a lot faster when you're done. There is actual comments in the header file. So right, you give it a function pointer to a call back.

PROFESSOR: Now, one sort of standard thing we decided we're doing is, we're not trying to write both 32 and 64-bit code in the classroom, we're just going to do 64-bit. So just one more level. In practice you would want to have stuff that runs both on 32 and 64, but for the most part it's fine if they just have it running on 64-bit.

ERIC: Right, so these by the way are the initial four points that are the input to the puzzle. I believe that's the problem set one. Questions about, yeah?

AUDIENCE: Can students submit more than one answer?

PROFESSOR: No. Nope, they got to decide what they're doing.

AUDIENCE: I guess I have a question about the answer.

ERIC: Yeah.

AUDIENCE: Is an answer that is probabilistically correct, I mean like, it will sometimes when you test it, you have it pass all the test cases, but mine isn't guaranteed to necessarily always solve all test cases is that OK?

ERIC: If they can conceal that to us, then I think they--

PROFESSOR: They deserve credit.

ERIC: Keep in mind that the other half of the problem set is to write test cases that will catch as many obscure bugs in the other students' code as possible. So they run the risk of competing against--

PROFESSOR: They lose extra points if they don't pass their own tests.

AUDIENCE: They each need to write some tests that will be as--

PROFESSOR: Yeah, so then they--

AUDIENCE: They exercise somebody else's.

PROFESSOR: --to somebody else's, yeah.

SAMAN AMARASINGHE: We are running all the tests against everybody else.

PROFESSOR: And then for the final version we give them the full regression suite of the whole class, so they have very good confidence, and we good confidence that they're handing in something that is correct.

SAMAN AMARASINGHE: OK, so you want to open up for--

PROFESSOR: For questions or comments.

SAMAN AMARASINGHE: Especially, I won't even mind the people who did it last time, last time to provide some of your feedback on how we can...

[BUZZING NOISE]

PROFESSOR: Don't want to do that?

SAMAN AMARASINGHE: Yeah,

PROFESSOR: I was trying to turn on the lights. Do we have more lights we can turn on?

SAMAN AMARASINGHE: Also, if you could turn these on, too?

PROFESSOR: Good. Yeah, so comments?

AUDIENCE: I want to say that's why I was a master last year, and I had a great time, but I figured I'd stay--

[INTERPOSING VOICES]

PROFESSOR: Yeah, could you stand up so people can hear you? Thanks.

SAMAN AMARASINGHE: Give him a mic. Give him a mic because I think it'll be good--

PROFESSOR: It'll be good to have these-- yeah. All right?

AUDIENCE: All right. Thank you. So the first thing I found really helpful was to take the reference solution and actually diff what the student provide because they often just provided a solution, and if you don't know what they started from, it was hard to tell what they'd actually done. So diffing the two, that was a good place to start.

And then I found incredibly helpful just to spend 15 or 20 minutes prior to actually meeting with them, looking at the code and writing like two or three or four things to talk to them about. And it might sound like it's going to be highfalutin and stuff, but it was like they're using their returning function pointers to local variables, stuff like that. It was really like what the hell the makefiles were doing, for example, is something that I think they really appreciated me telling them that didn't get talked about in class. Especially once I got a good rapport with them, they just ask questions, and I spent the whole time answering more practical stuff about the problem. We had a good discussion, so that's what I would suggest.

PROFESSOR: Thank you, that's great feedback. Yeah, as I say, don't underestimate-- I mean, most people learn the kinds of things you're saying, but somebody told them that, hey, these are practices that really don't make sense in a real software system.

AUDIENCE: People were, by the time we got to the projects, your standard pattern was highly representative, a bunch of flags you want to encode in a bunch of fields, right? And you've seen them a couple times, it's just a standard pattern.

PROFESSOR: Yeah, so one of the things you'll find in this first one is that they're very confused about the difference between unsigned and signed, and how shifts work, and what happens when, you know, how come I shifted this thing left 33 bits and I got a zero.

Why? I thought I was shifting a 64-bit value, but in fact, they hadn't done 1LL, they just did 1. And so there are things like that, where they don't-- very low hanging fruit, if you will, to talk about.

AUDIENCE: Did you tell them about valgrind by the way? Like, why would programs crash when they said they spent a huge amount of time debugging these crazy C programs when they're not used to the compiler not protecting them.

PROFESSOR: Actually, that's a good point because we were going to try to introduce-- maybe we should do that as part of the next lecture, spend a little bit of time on valgrind.

SAMAN AMARASINGHE: Yeah.

PROFESSOR: Because the memory checker there is really a pretty useful tool for them to get correct code.

AUDIENCE: I second that. That thing has saved my butt so many times.

PROFESSOR: So many times, exactly.

AUDIENCE: I've heard that, but more low-hanging fruit for valgrind is to always compile a -w all to just cache all the warning and declare all warnings as errors, and it just makes--

PROFESSOR: I think that we have that as a default in our makefile. Is that right?

ERIC: At least for the first project.

PROFESSOR: The first project?

AUDIENCE: I think they should understand that in industry practice, if you don't do that, you're a schmuck.

PROFESSOR: Right. Yep. Good, so that's good, and why, in particular, the valgrind-- we discussed that last year, and really actually, the upcoming lecture's exactly when we should do that.

SAMAN AMARASINGHE: Because there's also another tool called Dr. Memory.

PROFESSOR: OK, which makes sense, yeah.

AUDIENCE: One issue that I experienced a little last year was that people they specifically intend to write a little test for their code. They've mostly relied on a few runs by hand, then they wrote a progression producing things that they ran earlier. I guess that now you're actually encouraging them to actually have a test suite.

PROFESSOR: Yeah, well, now they get points off if they didn't write enough test that somebody else's code passed it that had a bug.

AUDIENCE: And in this case the testing run will be as far as the first turn in, and then the second turn in is just you have a version to run against and then just need to optimize.

PROFESSOR: Right, that's right.

SAMAN AMARASINGHE: So first, about the actual tests they have write a lot of tests because that will help, and the second, then you give them all the tests. So they will have a lot of test suite to run and then of course we have a lot of number of points we'll deduct of you actually fail those test. Because they have no excuse.

PROFESSOR: There's no excuse at that point, it's just a question of having--

AUDIENCE: What performance and measurement tools are you going to be using this year?

SAMAN AMARASINGHE: So, right now, we are going to give them perf because what we found was, the process of ECUT is so new a lot of performance tools don't have-- like OperaFile didn't work, because they haven't been ported properly. There's no port for it yet. So perf we are going to use, and we are thinking of giving them gprof in there. And then last year gave them couple Pintools we didn't ask them to write anything but for example, account number instruction or couple of tools they do that. Those are main tools you get, and also we need to go to Cilk, you'll do a bunch of Cilk tools in there.

AUDIENCE: Everyone last year, they were given VTune, I think it was, it was too hard for them to use.

PROFESSOR: Yeah, that was too hard for them to use because they have to use the machines remotely, and VTune and is an interactive tool, and so that just made it very complicated to do things. So instead we're going to go with a something that has textual output, and is easy, therefore, to run from command line and has all the advantages of running things that you get when you run things from command line.

AUDIENCE: So a logistical question so, meeting on MIT is fine, but where are there designated places to meet?

SAMAN AMARASINGHE: We will work with student to find places for them. For example, there are locations for PSS students to meet, and we are going to get them arranged. So if in the organizing you guys can create probably an online scheduling something. What's that website? Online is, online scheduling is--

PROFESSOR: Doodle.

SAMAN AMARASINGHE: Doodle, Doodle or something like that, and get their times and then they'll work with us to figure out the place to meet, and then they can send it out. One thing we will have to do is, between four students, because every project we are chaining the pairings, so sometimes we might end up in a way that, two of your students are paired together, and then we will tell you don't put them together in the same group, you'll basically have to mix them in the other way. If they are not in any kind of a pairing, then you can organize the four in any way you want.

PROFESSOR: Actually I don't even want of the four of them, I don't even want us pairing. That shouldn't be. But we'll see what happens. Barry?

AUDIENCE: Last year of course was in classroom downstairs and I just met them just outside the classroom. There's plenty of little space.

PROFESSOR: Also the first floor of STATA, I mean we've had a lot of people just using the first floor of this building. There's lots of cafe and place there, and so forth, to sit there,

get a coffee, and so forth. So some people just did something that informally, other people wanted a more quiet private place to talk. And I think that we'll help the students facilitate that.

SAMAN AMARASINGHE: And the key thing is, especially if any students are flaking out, for example, doesn't answer your email, or missed the meeting, or something like that, or late to the meeting, let us know. Because that's part of their grade, even though you're not grading them, if they are not participating in working with you guys, they will get points deducted. So we really want to know that, and we have tried to emphasize that they have to act, and actually do it right. So we really need to know that, if the students are not actually cooperating with you guys.

PROFESSOR: So I hope that this year will have a better-- So the vast majority of people last year had no trouble, but we did have a couple of people for whom it was one or two students who just made life a little bit miserable for their master, and so hopefully we'll provide a few more incentives this year to avoid that situation. But they're students and so they--

AUDIENCE: When do we find out which students are assigned to us?

SAMAN AMARASINGHE: Probably the next two days. OK, and then who's responsible for contacting whom? That's a good question. Who should we contact?

PROFESSOR: We decided that, didn't we, at the meeting? What did we decide?

AUDIENCE: This is what happens when you don't write things down.

SAMAN AMARASINGHE: You have to cover the old stuff.

PROFESSOR: Let's see, so last time I think we had it that the students contacted the mentor, and then--

SAMAN AMARASINGHE: What we will do is we will set up, we'll send the intro email to everybody, we'll send the intro email to the mentor, Cc the student then saying I am introducing you to each other. And then one thing you can do is immediately reply and, if you have

Peter Doodle Paul, or something like that. And then say, OK, look, do you know your students? So you can reply to that and hopefully they will reply back, and proceed. So when we send you the names you will also see the emails of students.

PROFESSOR: Are people familiar with doodle? Anybody not familiar with doodle? So doodle is a website where you can basically enter a whole bunch of times, and then people enter their names and mark off which times they can make things. It's a poll source.

SAMAN AMARASINGHE: Don't do what I did the last time, I marked off all the times that I cannot participate, so I was basically the exact opposite.

PROFESSOR: It's green and red, is what the colors they use. So green means you can make it, and red means you can't. So, anyway you don't have to use that facility, but it is a convenient one to use.

AUDIENCE: Can we expect students to be dropping?

SAMAN AMARASINGHE: There will be people dropping. They just started. This is also a group project and at 2:00 AM we got mails saying OK, I'm dropping the class, and we had to scramble trying to add a 3:00 AM to create another group, and so, Yeah.

AUDIENCE: You shouldn't take dropping as a reflection on our end. People drop out.

SAMAN AMARASINGHE: You have to know that.

PROFESSOR: You can if you want.

AUDIENCE: I had like three people drop

PROFESSOR: No, that tends to be more--

AUDIENCE: I had one student, it was awesome. So easy.

PROFESSOR: Yeah.

AUDIENCE: To get an idea of the mindset the students are going to be in this sort of a really

tough 6-170 type thing, or is this one of the easier classes.

PROFESSOR: No this is a challenging class.

SAMAN AMARASINGHE: So you take 6-170 that has about four projects. We doubled the number of turn-ins because each project has beta and a final, and some projects has two turn-ins in there. So basically, they have all their big projects going on, because they are doing the new project and they have submitted final on the previous project. And they're already completed project zero that was--

PROFESSOR: Just like real life huh?

SAMAN AMARASINGHE: We gave them project zero. That was built to to get used to the system and we can help them with the class, they have to finish this one. So this class just screams excess very, very--

PROFESSOR: Yeah, and of course they're students, some of them have learned you put off things to the end, and haven't learned the lessons of well not always. So as I said, as a group they're relatively immature when it comes to some of these things, but there's a large fraction of them who are incredibly mature about these kinds of things. So our job is as lecturers in the class is to make it so that the students who start work early don't get penalized.

And that's probably the hardest thing in the software related class because, the classic thing is that they want to get started, they find all the bugs right? And they waste their time doing it, and if you do it later, then the TAs have worked all the bugs out. So one of our challenges is to make it so that students who start early are rewarded by being able to have more time not wasting their time.

SAMAN AMARASINGHE: And today was the day that they turn in the project and by looking at the class, you realize they haven't gone to sleep, and I asked and most people haven't slept in about 30 hours. They have been up entire last night and an email conversation when I went to sleep there was an email train going. Every five minutes, something pop up. When I woke up, the train was still going, and amazingly, there were a couple of TAs up and answering all through the night, and that kept continuing.

So these kids they can be flaky sometimes, but they're amazingly hard working. And this is only one class they're taking. They might be taking four classes like that.

And I have seen people scheduling their schedule as by hour, scheduling and they say, OK, look I can do the problems set or I have one hour of sleep. And a lot of times that's their thing, and then sometimes they triage. That's why sometimes they miss classes, they realize, OK, I can do the problem set, I can sleep, I came do that. So I think these kids work amazingly hard.

PROFESSOR: I do think it's helpful to also counsel them on, hey, you can't do everything on a death march. You got to learn at some point what you're capable of, and manage your own time resources and so forth. Because the joke at MIT is, classes, friends, sleep, pick two.

The other thing, let me just say, is that despite the obvious sleep deprivation we saw in the class today the number of students who came up after class so charged about working on this first project was really quite amazing. They were all very, very excited. And very excited we also gave them the initial feedback on the performance, and you could see, there's one group of students who clearly got it, and then there's another tier of students who didn't get it quite this much, and then there's the ones who their code has errors and stuff. There's no doubt that there are a lot of students who are very charged about this. This is a fun class but it's a hard class.

SAMAN AMARASINGHE: I think you'll find this is true. They will be, probably tired, but they will be soaking up this information like a sieve, and sometimes I found last year's-- After the looking at the first set of code, some of those the masters were really upset. How can you teach you guys, these guys write this kind of bad code, but when they realized after they said something, this is getting into their heads. I mean, they're improving they're actually changing.

These guys are very smart, and they're looking for you guys for-- They're basically sponge, really empty sponge. They're going absorb a lot in this class.

PROFESSOR: I have a 2 and 1/2 year old at home, and it's amazing what the progress is she's made in six months. She talking now in complete sentences, and so forth, it's just really amazing how how she does it, and these students are like that. They start out and you say, oh my God, they have no skills whatsoever.

By the end of the term the progress that they've made is really astounding. You just say, gee if you could just keep on that trajectory your whole life. But really, they really are very smart, and they come up to speed very, very fast, especially given good counseling. The main thing that they need right now is just the attention so somebody actually can look at their code and say, you know you did it like this it would be better code. Any other questions or issues? This is lots of fun. If you run into any troubles, we're right here, just send us email.

And as I say they're young and immature, but then you get to see them grow, of course some of them will flake out on you, but the vast majority, you'll see, are really very dedicated and well meaning. They really want to do a good job in the class, and really, really find the class very exciting. Our numbers, in fact, almost doubled for last year. So the reputation got around that this was a good class to be doing.

And the other thing is, right now what I really like about the class, the students right now compared with say 10 years ago. So 10 years ago we had the Internet boom, and there were all kinds of students taking computer science because they thought it was a ticket to fortune and so forth. And now we're back to what it was pre-2000, 1998, whatever.

Where the vast majority of students who are taking the class really enjoy engineering. They really enjoy the software. They really enjoy the engineering. The fact that you end up with a pretty good job and opportunity sometimes to do entrepreneurship, and opportunity to explore things, it's sort of like, oh that's a fringe benefit. But what they're really after is just the intellectual excitement and so forth.

So it's really quite fun to work with the students and quite rewarding. Really very nice type of class to have right now, where the students taking it because they're

actually interested in the material, not because it buys them a bread ticket, meal ticket, whatever. Yeah, question?

AUDIENCE: The standard, is the coding standard in the reference, the standard is to consider for them provided?

SAMAN AMARASINGHE: I don't think we have set up a standard, per se. What we want them to do is kind of create their own style. It's like in a writing class, you don't give them a standard writing guideline is to follow this one, you don't constrain them to that. We want to give them the opportunity to grow into their style, but kind of guide them, so that they don't make the classic mistakes.

That's why we need you guys, I mean, if there's a standard, we can come up with a standard, create a script to actually check the standard, and then say, OK, thou shall follow the standard. And you can do that and everybody will hate it. You can't make that bad happen. One thing you want to do is make them lifelong good coders, not something they realize OK, I need to do this to get this grade, and I'm going to abandon everything and convince myself never to that again.

And we have had sometimes, like when we needed all this software engineering. We went through things like thou shall write all the preconditions and postconditions, and everything will be graded. And people did it, but half of them got into them and said this is good things. Other half, they didn't realize why they're doing it, they just hated it. And I don't think they left assuming I would never do this again.

So the key thing here is, we don't want force something on to them. We want them to come through a natural process, and I think a lot of them, can, when they understand what they're doing, they'll develop a really good kind of practice.

PROFESSOR: It's also the case, I think, that coding standards vary culturally from place to place, and the what people expect in one place is different from what thing. So being too prescriptive means, as Saman is saying, that it becomes a grammatical thing, rather than is this code organized. The real thing, coding standards are very important, especially in large software organizations of course, so that everybody can tell

what's going on in any particular piece of code without having been the author. But in this kind of thing where we have something that's relatively small, what we're mostly interested in is the organizational aspects of coding style, as opposed to the grammatical ones.

SAMAN AMARASINGHE: So also don't impose your standard and say, my company this is what we do, I want to follow that. Just figure out and kind of guide them towards something while they're maturing, and try to make it better. That's the thing. That's why you cannot have a cookie-cutter model.

PROFESSOR: And why it's good to have senior software engineers who understand what's really important, as opposed to what might be required in any particular individual situation.

SAMAN AMARASINGHE: One other topic I want to add is sometimes when you have two students, there might be a case that one person might be very talkative and dominating, and one person might be just a listener, and so you have to keep an eye on that and make sure that you give the other person opportunities to talk, and because it might be just like that. Encouragement and stuff like that, so make sure that you get both of them to talk.

And also the are supposed to be responsible for all the code. You should not let them get away by saying, oh my partner wrote it, so I don't understand. So that's not a good answer. So get them to actually go through that, and if they don't come prepared tell them that, and you can give them some feedback, as a next time can you please do, this and this. Because they also don't know what to expect, and we give them a little bit of guideline, but it'll be good, also for you, the first time around, if you find some mistakes also for the next code review you can encourage them what to do, how to prepare and what to do.

Don't give them too much work, because we're already giving them enough work. Don't ask them to write extra documentation, but you can say it would have been nice if you had this level of preparation, if you could have explained to me, or given me a summary, or something like that. So something like that is something you can

expect.

And the other thing is if you had to figure out how we want to do code review. So, for example, you might want to bring your laptop, or ask them to bring a laptop in there, if you want each other to look at the code on the computer, so ask them to bring the laptops or something. So think about the logistics of how you're going to do that, and that'll be useful.

PROFESSOR: And I think any other sort of expectations you have, in your email to them the first time say, here's what I'd like you. I think it's helpful to say I'd like you to come prepared to give me a five minute update on how you did each of the problems, for example. We will be telling them that, but to have it come directly from you means-- The difference if I say go do this to that person, versus you're coming to me and expecting it. Good.

So, I want to once again, I can't thank you folks enough for being willing to give your time. Time is the stuff life is made out of, so I really appreciate your willingness to volunteer your time for these students, and I really think you have a dramatically meaningful impact on the students. The students may or may not fully appreciate that, OK, but Saman and I really appreciate that, and we will make sure the students appreciate it.

AUDIENCE: Yeah. I just have one. What what are the guidelines for a contact outside of the code reviews with the students. I mean are they, can we have them email us with questions about the process and just kind of general guidance

PROFESSOR: It's your level of tolerance we don't want you to have to commit more than about 20 hours across the term, because then it becomes onerous. But if you're willing I think it's completely open ended, in terms of how much else. I mean it's one thing if you go into solving their problems for them, that we don't want you doing, but when it comes to, especially the quality, and non-metrical aspects of programming, the more you can give them the better, OK?

Because it's only making them be better programmers. And I think the fact that

there may be some unevenness in that across the masters is irrelevant. Really, I do not want to hold back some students from getting good mentorship because somebody else is not getting the same level of attention, et cetera. But really, no expectation there, on our part, to do more than the code reviews themselves.

SAMAN AMARASINGHE: So, that said I want to make sure that you aren't answering their question, at some point I don't want to be a crutch for them. For example, if they say, OK, before prebeta, they say OK, this is my next problem, can you on a little bit, I didn't understand. If you have that, so they're trying to get too much into that level, kind of making you their programmer, or something like that, just let us know.

A lot of these things, there will be a lot of cases that arise that we haven't thought through. The students are ingenious enough they will find a lot of loopholes, a lot of kind of interesting things that we never thought through. So if you find something that you're not feeling comfortable just send a mail and say OK, take a look.

PROFESSOR: It really is, if their questions are about course content it's not appropriate for you folks to be dealing with, really. OK, that's what our TAs are for, and course staff. If it's about coding style and engineering practice, and even time management, or anything that's sort of the non-technical aspects, but things that have a strong impact on technical, that yes. Anything you can provide there is great.

SAMAN AMARASINGHE: So if you find anything that's outside the norm of what we discussed if you feel a little bit uncomfortable, just send email. Send an email, and say, hey, can I do this? Is it OK, for me to do this? And then we can always provide guidance, so we are responsible.

PROFESSOR: --do something and say after the fact, oh, it went past, let us know. There's rarely anything you can do on one shot basis that is heinous or unrecoverable from, or whatever, so.

SAMAN AMARASINGHE: It is always good, I mean that's why-- since we see everybody able to and we have done these kind of situation before. We can just easily tell you what.

PROFESSOR: But, mentor them, for sure, mentor them.

SAMAN AMARASINGHE: I think a lot of times, when you talk to students last time they got a lot more information out from the masters, other than just the coding style. I mean, they talk about what is it like to be an engineer in a company, what are the things? And so they get a good feel. I mean, some of them might not have done a internship in a company. So they don't know.

I mean, they're just green, and talking to you guys they get a feel for what it's like. And also from a non-marketing situation too. I mean you might have ulterior motives to market your company to them, and stuff, but this is not that they're going into a job interview.

So, for a lot of them it's rare. I mean they might not have retinues of friends that they can go and talk at that kind of casual level. Most of the time they're on guard because they're on an interview or they assume that information coming to them will be sugar-coated, and here is a way that they can get-- I think it's very good for them.

PROFESSOR: Eight minutes over time.

SAMAN AMARASINGHE: OK, I think we should.

PROFESSOR: So, any last final comments? Saman and I will hang around for a few minutes, for anybody wants to chat more one one. And one and once again I thank you for all the time that you're planning to put into the 6-172, thanks.

AUDIENCE: Thank you.