PROFESSOR: So today we have from EA, Timothy Cowan's going to give a lecture on development best practices from Electronic Arts. This is about the time of the class where we start doing hour long lectures and then the remaining two hours is open for your work time.

It's really going to start being like that probably not next Wednesday, but the following Monday is when that's going to be a pretty stable time. So when you're working on project four, you're going to have a lot more in class time to work in teams. Hopefully you've been taking advantage of the class time to work in your teams for project three.

Today we are doing play testing. Has everybody already prepared for what they're going to do for their play test today? Do you already have questionnaire, survey, observer notes, things like that? I'm seeing some blank faces. That's OK. We're going to give you 15 minutes to set that up.

So we're going to do lecture, take a short break, then you'll get 15 minutes to prepare for the focus test. We're asking you to set up three workstations for the test at all times. I'll go through the details when we start that up. But you'll get a little bit of time to set up for play test.

Remember, we're asking for two focus test reports for project three. This is going to count for one of them. The other one you need to do on your own time or have already done.

Grading is practically finished for project two. And now it's just entering it all into the Stellar form, and you'll have all that by tomorrow night. What I can say just in general for grading, project two looked pretty good. Most of the games completed all of the requirements.

The main requirements that we saw that were lacking in some of the games were really the legal requirements. Putting your credits, putting your names on the game. We asked for that. So please make sure you're doing for project three. Even if it's not in the game itself, it can be on the full sheet, the full HTML code for the game. So make sure you're putting your names on your games when you're turning them in.

And also if you're using any outside code, any outside assets, that should all be listed as where it's coming from and what license you have for that. Phaser is MIT licensed, so there wasn't anything special for there. But if you're using Unity, take a look at Unity's license requirements. Make sure you're fulfilling them. If you're using hacks or Flixel, make sure your filling those requirements. I believe we already gave you some comments on the post forms. But for the most part, those also looked good.

For project three, here's the cheat sheet for product three's postmortems. You've got a page to write a postmortem. Make half of it about your team practices and processes. The other half about the game and the design. That's really what we're looking for is understanding how your team organized itself, what communication problems you had, how you solved them, and even more importantly, what are you going to do in project four. Because in product four, you're going to have eight people on your team and problems are going to show up more often. So hopefully you're coming up with ideas and ways to be prepared for those kind of things.

Any other comments? Otherwise for the other turn ins for product two, task lists were good. Backlogs were acceptable. The focus test reports for the most acceptable. We give you some comments about your focus test reports. We gave you some comments about your design change logs.

We also are giving you a big chunk of comments about the game design. It's going to be labeled not part of the grade. So like I said, it's not a design class, but we do want to give you some feedback on your designs. So we can help you out with future assignments. That's it. Any questions about anything? Great. So I'm just going to hand it off to Tim.

TIM COWAN: Great. So can everybody hear me? Sound good? You can just raise your hand in the back if you can't hear me. Good? All right.

So let me start off a little bit with just introductions. Who I am, my background, where I came from, why I'm here, maybe. I'm actually originally from Oklahoma. I was born in Tulsa, Oklahoma. And went to Oklahoma State University. And going to university, I actually had no idea what I wanted to do. Kind of just sort of landed at one of the in state tuitions for university. But I knew my dad was an engineer, a programmer specifically, and I kind of thought that I wanted to go do what he did. So I went into university, was going to go be trained to be a programmer. He actually gave me advice to go work on more pure engineering. And so for my undergraduate degree, I was actually focused on mechanical engineering. And I stayed at

Oklahoma State University and went on to aerospace engineering.

Did graduate school. Was in pursuit of a Ph.D. with ultimately the goal I wanted to be an astronaut. I didn't make it there. I made video games today. But I did get a chance to through graduate school work on research for Dryden Flight Research Center in California and actually spent about 10 months there as a visiting researcher while I was working on my Ph.D. And what I did for them was what I'd say were flight simulators. But not that kind of flight simulator. It was more this kind of flight simulator. Computer simulation of aircraft. I did a lot of both the simulation of the aircraft and generating visual representations of the simulations.

And ultimately out of all that work I did finish my Ph.D. and sort of officially became a rocket scientist. And so then I had to decide what I wanted to go do. And my plan really was to go work at Lockheed, design aircraft, work on flight simulators for them. That was basically my life's mission coming out of college. But that didn't quite happen either and I ended up at EA Sports.

So instead of working on flight simulators, what I've done for the last 14 years is work on NFL football simulators. A recruiter from EA just basically blind contacted me from my resume posting on Monster.com and they brought me down to Austin, Texas to work on my first video game, which was Madden NFL 2002 for the PC. There was a small team in Austin of about six people that built this product off of a shared code base from the PlayStation to Xbox and Game Cube.

And I spent about 18 months in Austin. Built two versions of Madden PC in that time, until they eventually moved me out to Florida, which was where Tiburon Entertainment was located, one of the studios that's part of EA and part of EA Sports. And I've been there, really been with Tiburon for I guess it's counting 13 years now. And in that span one of the things I'm most proud of is I built 14 versions of Madden in that time. And I can say, with the exception of Madden 15, because my roles changed, I've contributed at least one line of code to 13 versions of Madden over the last 13 years.

Things are pretty different at Tiburon from when I started in 2001. There were about 200 people at the studio when I started. There are 700 at our location today. We work on three different products. And I'm the group technical director for the studio. So I help make sure that all three of the products we build succeeds and sort of monitoring engineering and technology development for the studio. Our three products are obviously Madden. We build NBA Live and

we're working on the next generation of PGA Tour Golf, which will be out this March.

So that's my background. A little bit about my role. It is very much it is an engineering leadership role as a group technical director. And I very much don't write much code. In fact, I kind of joke with the other engineers that the coding that I do is PowerPoint these days.

Let's talk a little bit about Madden. Set a little bit of context about some of the philosophy and best practices that I hope to a kind of pass along today. Madden in North America has been a we'll call it a top five product for at least the last 15 years, typically is a top three product in North America. Generally the first person shooters are the big sellers, especially in North America. Sometimes you've got some of the action games like Grand Theft Auto. But depending on what's being released at any given year, Madden's always a number two, number three in North American sales.

And you can see FIFA top 10 North American sales. They're a different sort of a sports product. That is the big sports product but they have global appeal, because the sport has global appeal and it sells in a lot of markets that Madden does not. But as a North American company, Electronic Arts, the number two title in North America. In fact, on that last chart, the number one title in North America that EA sold is Madden.

And it's very important to EA. It's very important to EA stock price. If you look at any sort of financial reports or just even news briefs on financial listings for EA, you'll often see that the little blurb at the bottom where they kind of describe who Electronic Arts is, Madden's almost always listed as Electronic Arts, developer of products such as Madden, Battlefield, FIFA. Madden is very important to EA. And the success or failure of that product is very important to the public company's stock price.

If you look at Madden development, given its importance to the company, given the size of its sales, it's a big team. There's over 200 developers on staff that do direct development on Madden. That includes engineers, artists, designers, producers, product managers, quality assurance testers. And there are additional developers that contribute through central. Technology We're a very large organization that contributes in a lot of different ways. But sort of at our studio we call it 200 people doing direct dev on Madden.

10 million lines of code, give or take a few. That's physical lines of codes. So it includes white space and comments and those kinds of things. But a lot, a lot of code to look at across 150,000 plus files.

And we do development distributed across four different locations. So we've got primary development in Orlando, but we also have remote development in Austin and a contract company that we work with Nova Scotia. And Vancouver is the sibling studio for Sports, EAC, who makes FIFA, NHL, USC. They contribute a lot of central technology that helps ship Madden each year. So there's really sort of four locations that we're distributed across.

Really the message here is Madden's big. It's a big seller. It has big importance. And it's big large scale development. Large teams, large code bases, large complexity.

And the other characteristic about is kind of important to sort of understand sort of the nature of this project, this big project that also iterates at an iterative cycle every single year. So our sports products come out at the start of the sports season every single year. And it's critical that Madden comes out on its date in August when the NFL season kicks off. Our gamers expect it. Our shareholders expect it to sell. And if you miss that start of the NFL season and were to, say, slip your ship date, that means a lot of lost opportunity in terms of delighting our gamers.

That's very different from other types of game development. Like certain games, they're done when they're done. And even within EA we do this a little bit. This was one of the announcements we made about Mirrors Edge. That's not Madden. Madden can never miss its date. And it has come out in August for the last 25 years consistently and reliably.

So given that context, what I want to talk about a little bit first is just some philosophy. And this is some philosophy I've come to over the past couple years about what I think makes a good engineer at EA and specifically on Madden and our sports products. What some of the mindset then things that you may need to break out of in order to be successful doing development on the big title that's got to come out on a particular date.

And when you are big title and a big team and a complicated project that has to hit a particular date, the first thing that happens is we go add a lot of really complicated process. Because there's a lot of moving parts. Everything's got to get tracked. And if anything slips, then we're not going to hit our date. You add a lot of checklists and you check things off. Hopefully get everything checked off.

But the reality is there's always a lot of work to do. We've got 200 plus people doing a lot of different types of work and you only have about one year to get work done. And we want to do

and put as much as we possibly can into that product every single year. Because if nothing else, I know I'm very sensitive to our sports products.

Because they come out every year, we tend to get beat up a little bit over just a roster update. They haven't done enough. There's not enough improvements over last year's product. Well we do a lot of work and we want to do a lot of work to make sure that every single year we're putting out the best possible product that we can. And it's not just us. Also our gamers really want-- they're giving us a lot of feedback. We want this, let's get this feature in, Madden would be better if you just did this.

So there's a lot of work and our kind of human nature is let's do it all. We're going to get it all done, we're going to ship an amazing product, and we're going to get it all done on time in August. Well if you take on too much and you try to do everything, disaster happens inevitably. And one thing we have to remember in sports is development's a multi-year journey. It's a path.

Most game products, especially if you're looking at a open world action adventure game or first person shooter, they tend to have two if not three or four years of development. We kind of start to think about Madden development in that same way where it's a three year process with incremental releases along the way, where the game just keeps getting better and better and better on a three year roadmap. And it's important in this sense because I think all game makers want to do everything, want to make the game as amazing as they can.

But in sports, you really have to embrace iteration. And that means kind of finding the right balance between doing everything, doing everything right, and especially if you're an engineer remembering that sometimes you have to balance between getting everything done right and maybe using some duct tape and kind of hacking in a quick solution. Because you've got three years to make sure that you're getting things done on that road that you're on.

So another idea I'd like to talk a little bit about, again, just more philosophy about engineering. I talk about the difference between science and engineering. And I'm going to sort of give the extreme examples and maybe overgeneralize things.

But one of things I talk about to my engineers is, if you look at science, science at its core is about the pursuit of knowledge. The idea is king. You're going to try to figure out how things work. You don't know what you're going to do once you find it out. But just the fact that I know how that works, I know this idea, that's what's most important.

Engineering, on the other hand, if sort of like the best engineer, if you're just the extreme engineer, the thing you get the most excitement out of his taking that knowledge from the scientists and figuring out how to apply it to solve practical problems. And cost effective solutions is definitely one of the keys in terms of that well rounded engineer that goes in and is figuring out how to solve a very practical problem based on a body of scientific knowledge.

It's really important for engineers remember that there are solutions to even the hardest problems. If cost is a factor, you often find that engineers will tend to say this type of problem's not possible. But what they really mean is it's very, very expensive. But engineers need to really be focused on, how do you break down a problem and come up with a solution to that problem? And sometimes it's duct tape. And for the *Star Trek* fans out there, remember there are no unwinnable scenarios in engineering. There is no *Mission Impossible.*

I've had engineers on Madden in particular where Madden in the early Xbox 360 generation actually ran at 30 frames per second. And there were a few years there where we were told it was impossible for Madden to run at 60 frames per second. And so in a leadership position, a lot of your job is to kind of break through that this is impossible and start to ask questions on how can we break this problem down and come up with some cost effective solutions that will actually allow us to solve this problem.

And at the end of the day, you're not going to solve problems if you don't attack them with enthusiasm. And so my encouragement is really to challenge the impossible. Everything is possible. It maybe might not be possible today. It might not be possible with the money you've got in your wallet. But given enough time and money, we can do anything. And every problem can be solved. At least at the core, a good engineer really needs to believe that.

So let's go on next idea. And I want to dispel the myth of perfect code. Because this is one of those things that engineers coming into EA, really coming into any job, have this myth that there is perfect code. Because engineers love perfect code. And they really love to argue about perfect code.

Even the best engineers in the world, this is ingrained into every engineer that not only is there perfect code, I know what that perfect code looks like and I'm the only one that writes it. But the reality is, mistakes are always made. It tends to be the first time you write code, especially if it's the first time you've written code for a problem you've ever solved before. You're going to make mistakes.

And I do very much fundamentally believe that kind of at the core inside that you learn more from mistakes. Hopefully you guys have heard that before. And if you never make any mistakes, you're not learning. And if you're never taking on really big potentially impossible challenges, you're not learning. You're not stretching yourself, you're not growing. So it's important to remember that mistakes will be made. You're going to learn from those mistakes, and you know what? Your code will get better as you make those mistakes and refine it.

So remember, there is a software development life cycle. And in fact, a lot of times when you're arguing about perfect code, it's very early on in the cycle and you don't even know what you don't know because that code hasn't been through that software development life cycle.

There's almost no guarantee that there's no bugs in this code that you're writing. And it really isn't until you make it through that software life cycle that you're going to understand what those bugs are. Towards the end of the project where you have this perfect code and all of a sudden now it's been all the way through testing and certification and now you see, oops, I made a mistake.

And when you tie that into our sports iterative cycle where we're embracing iteration, taking on the impossible, one of the big things that try to impress on people is try to value less the perfect code and start to think about proven code. Proven code that's been through that software development life cycle, has had the bugs shaken out of it. Even if you didn't write it, that code is proven and will have less bugs typically than code that you go right from scratch.

And then the next idea, and this is the last one before we can get onto the other section and get out of the philosophy stuff, is on a big team, on that iterative cycle there's a lot of tasks. There's a lot of work to get done. Everybody's given their tasks and they have to finish their tasks.

They've all kind of got this little piece of a puzzle and they own their piece of the puzzle. They've been given this task and they're going to get that piece done. And it's going to be perfect given the context that they've been given on that task. The trick is there's a lot of dependencies. And it's often very hard to go see, how do all those pieces together?

And when you've got this kind of situation, you start to end up with you've got these features, which are actually a collection of tasks. And those features don't really have an owner. All the tasks have an owner. Someone's doing the tasks. But what about the feature? Does the

feature have an owner? Does anybody know how all those pieces are going to fit together?

And if you don't have someone owning that feature, you end up with unfinished projects. Or potentially you end up with not quite what you were looking for because you didn't really understand from this task how it all fit together and was going to ultimately deliver whatever it was that we were looking for.

And so it's super important when you get immersed into a really big team and you're hand and a bunch of tasks to pick your head up every once in a while and make sure that you're focused on that feature. And potentially you're asking, how does this task that I'm working on fit into this feature?

Because if you've got that context, you may even realize, hey, my task actually isn't going to help me deliver on that feature. I'm not working on the right thing. As an engineer with a technical background, I'm building good software, functional requirements, et cetera. You really need to be focusing on that feature. Because your task is just a means to delivering on that feature.

So in summary, just a little bit of philosophy type stuff. Embrace iteration. Challenge the impossible. Value proven code and focus on features, not tasks. And that's kind of my top four. It's easily a top 10 list. The thing is to kind of think about in terms of coming into a big company, working on a big team. Some of the philosophy that you kind of need to take with you in your approach on integrating with that team and being successful.

Now I want to talk a little bit about more on the process side, engineering quality. How do we as a engineering organization at the studio help make sure that we're delivering a quality product. And game development's notoriously not very formal. We are game developers, after all.

The challenge, though, when you're not terribly formal in your development methodologies and your development practices is success starts to rely on individual heroics or individual heroism. And that's a bit of a danger zone when you start talking about shipping a quality product that's big and on time. Because quality's never an accident. I love this quote. It's never an accident. It's the result of intelligent effort.

Now, before we talk about processes for this, let's decompose quality a little bit. Because this kind of plays into some of the thinking on our approach to quality. And really, it's going to focus

on the different aspects of quality. There's perceptual quality. Is the game fun? Is it beautiful? It's how the gamer experiences the game.

There's functional quality. That's kind of, does it work? An engineer is given a spec and writes code and it either does or does not do what it was intended to do. And then there's structural quality. That's the underlying, behind the scenes quality. It tends to be the type of quality that engineers care about the most. That's kind of the code quality space. Is my code perfect?

The top two perceptual quality, functional quality, those are things that the gamer is going to see in your product. Structural quality is not something that the gamer will actually ever see or notice. But it is something that will hold up your project. If you don't have the right level of structural quality in your product, it will prevent you from delivering that product.

Now ultimately, at the end of the day we're probably most interested in that perceptual quality. Because that's the one that's most going to resonate with the gamers. Is it fun? Is it beautiful? And we kind of like it meta critic as one possible measure of whether our game's being perceived as a high quality. This is basically a reviewer sits down, plays the game, or even if you go look at user reviews. A gamer sits down and plays the game. And then that game experience gets processed through their filters and they perceive it to be at a certain level of quality.

Now, functional quality is inevitably going to play into this. But we generally see, I think, with meta critic is a lot of the functional issues may not be seen because it takes hours and hours and hours of playing the game before those maybe surfaced. And it tends to be, especially with meta critic, you spend four to eight hours playing the game, you process that experience, give it a score.

So I tend to kind of argue, at least when we're talking internally about quality, only the absolute worst functional issues are going to show up and play into your meta critic score. But the gamer who plays your game for 100 plus hours is really going to notice a lot of those functional quality issues.

We also have been talking recently about kind of a spectrum of quality. And this is kind of tied to meta critic. And I steal this from one of our executive producers where from 0 to about 69, those games are sort of just foundationally solid.

If you're anywhere below 69, sort of the upper end of that spectrum. You get to about 69 or 70

by being foundationally solid. Anything below that, you're buggy, you're broken, you've got a lot of issues. And from there, quality kind of builds. If you're buggy, broken, and have just glaring problems, you're never going to get to that next level where you've got good function. It functions and if you have a competitor in the space, it functions at least as well as the competitor. And that's kind of that 70 to 79 range.

But what we really want to do is we want to get up above that. This is interactive entertainment. We're trying to really delight the gamers that play our game. And from 80 to 89, we kind of start to talk about, all right, that's the point where your game is getting immersive. Especially for Madden, it's starting to be very realistic and they feel like they're there on Sunday watching a game and they're sort of immersed in the experience.

And then even above that, which is where we really want to get to is 90 to 99 or even 100. I don't know if it actually goes to 100. But that's where now not only are you immersed in the experience, but the game is evoking emotion.

So when you get up into those upper bands, that's where perceptual quality really starts to come in. I'm going to talk a little bit more about that foundational and that functional quality, because that's what engineers and development directors are ultimately paid to solve.

And so the question is, on a big project with a lot of complexity that's got to hit a particular date, inevitably need to put a lot of process in place to try to do make sure that you're putting out that quality product or at least at a minimum were able to hit that foundational and functional quality so that the designers and everyone else can help push us up into those perceptual realms.

The first process is code review. So let's talk a little bit about code review. Because engineers love code review. Especially because their code is perfect and no one needs to look at this code to figure out whether or not it should be allowed to get checked in. At EA and on Madden in particular, we do have a blanket policy, all code gets reviewed. If you're going to check code in, it's got to get reviewed. We even in this day and age where data is a huge part of development. Actually there's a lot of data that gets reviewed before it gets checked in.

But code review is sort of the primary function where all right, you're going to check in code. It needs to get reviewed. And the goal there, let's make sure there's no bugs. Get a second set of eyes, have someone look at it. Shake out all the bugs. Because the second set of eyes will find the bugs before it makes it to QA.

There was an interesting study recently, Rebel Labs. They did a survey and asked development teams, how many of you do code review and how often do you code review? Code review is very prevalent coming out from this industry survey. I think it was 76% of teams do a lot of code review. And their ultimate goal with this survey was to figure out the impact of code review on quality. And quality for them was defined by lack of bugs. So the functional realm of quality.

And their study produced kind of an interesting result. They were actually looking at two factors, quality and predictability. Both super important for sports. And they found that code review really helped with predictability. It indexed well.

You could see if you did no code review, there was an 11% or 10% swing between no code review and always doing code review. Much less influence though on quality. They found that there was not actually not much of a difference between teams that did code review and didn't do code review in the span of quality. Again, quality is measured as bugs.

AUDIENCE: What is predictability?

TIMOTHY COWAN: Predictability is able to hit your schedule on time.

Now, that's not to say that we don't find bugs in code review. But I think that's interesting because it kind of aligns with some of my experience within EA that, now, don't get me wrong, we put code review in place because we wanted to find bugs early. And when I started at EA we did not do any code review.

In 2001, again, this is kind of individual heroics, pretty informal. On a 200 person team, you've got to do code review. And we do want to find bugs. But it's actually a little surprising that we don't find many bugs in code review. And I see that at our studio as well.

How many people here know C++? All right. Awesome. Let's play a little game quick here. It's called find the bug. You guys are code reviewers. Raise your hand if you can spot the bug. There is a bug here.

AUDIENCE: [INAUDIBLE].

TIMOTHY You can't see it? OK. Is it the contrast or the lights? Anybody that can see it spot it?

COWAN:

AUDIENCE: [INAUDIBLE].

TIMOTHY COWAN: Good. Good. That is the mistake. It's a cut and paste error. So someone moving fast. And it tends to be pretty easy not to notice this. It's just kind of hidden in a block of code. It actually was intended in this case. It's just kind of following a pretty simple pattern. They should have been copying PURL into the STR web URL.

This was code that was written, tested by the engineer that wrote it, I presume. Code reviewed, checked in, and made it all the way out to QA. I don't know if an actual bug manifested, but this code did make it through code review. In fact, all this code we're going to look at made it through code review. How about this one?

AUDIENCE: [INAUDIBLE].

TIMOTHY COWAN: All right. Yeah. That is kind of the trick with code is it's kind of tiny text.

[LAUGHTER]

It's probably the sunlight. People in the front row can see it? No? I guess this game's a little less fun than maybe I was hoping. In this case, you've got two different files and this is a function call. And there's one line of code where they're using the function.

In the second file, if you're doing a code review you've actually got to-- in this case, someone added a line of code. And when you get that code review, you get just that file with a little diff and you see their line of code that they added. You won't actually spot this blog unless you open up the other file that they didn't edit and go look at that function and how it's used. Because they've actually got the two parameters reversed.

They're both bools. Track actives versus track passives. And the function they're calling has those two parameters reversed. Again, written, tested, code reviewed, checked in. I don't know if this one's readable at all. I kind of have to skip through this. In the interest of time, I'll go ahead and just skip through these. Because I think they're all somewhat fuzzy and small text.

In this case, if you don't know C++ or don't know the syntax, with that indentation there's an if

statement there. With the indentation it was probably expected that that code would only get it executed if the if condition was true. They missed their curly braces. And that that indentation tends to, if you're reviewing the code, kind of draw your eye. It's like, yeah, that code looks OK. You're kind of inspecting it. And I'm sure that they knew what they were doing. And you kind of missed that the braces aren't there. And this one's maybe-- no.

So in this case, their if statement, it's actually a constant number. This will always be true. And they probably actually meant to check if that value was equal to some particular value. But they actually just put the mask right into the if statement. And that will always be a number greater than 0, which will be true.

And last one. This last statement here. This last statement doesn't do anything. They're just assigning one variable to itself. Again, really easy cut and paste errors.

And I think the surprising thing in a lot of cases, again, we've got this process. We do find bugs. But you don't find all of them. And a lot a really simple ones, things that after the fact, when you get a bug report from QA and you go look at that and you're like, how come no one caught this? How come the person that wrote it didn't catch it? How come the code reviewer didn't catch it?

I'll tell you, in some cases it's not even just one person code reviewing, but you may have five people that code review it. Again, we're big, multi-disciplinary. Sometimes one check in may take five different subject matter experts where I need a rendering engineer, I need a physics engineer, and I need a UI engineer to review this code. And five different people review the code and no one catches the problem.

But there are other reasons for code review. And this is one of the things I'm pretty adamant on. Because when that report comes out that says, code review doesn't really find bugs. And in fact, if you don't do code review, you're not all that bad off in terms of finding bugs. I say, wait a minute. We're not going to stop our code review process. Because there's other reasons for doing code review. And finding bugs is actually, I would call, my fourth reason.

Really, code review's about monitoring structural quality. And that's why that report says if you do code review, you get better predictability. Because your structural quality is this underlying quality that's going to help you be successful. It's going to help with your predictability. It's going to help with knock-on effects, where if you write code that's consistent, reliable, robust, and file certain programming patterns, you're going to end up with a better result in the long

term and you'll be more predictable.

It's also about knowledge sharing. So on a big team of 150 people where everyone's got their own little task and they own their little task, we need to make sure that the whole team has some insight into what all the other developments are doing. And if you're out sick or you win the lottery and stop working, someone else at the company's has maybe done some code reviews and has seen what you're doing and has learned from that and knows about how that feature, how that task works.

And then it's also about education. So if you are the best engineer on the planet, you've been writing code for 14 years, and you've learned from all your mistakes, and you write the most perfect code, having a new engineer who just came to work at the studios, is writing their first line of code ever review your code teaches that person something.

And then the last point is obviously catch bugs. We do want to catch bugs through code review. It's just it's not the only thing that you get out of code review. And really, static analysis and rigorous testing is the best way to catch bugs. And in fact, all those examples I just went through are caught by static analysis that we apply to our code base every day.

Static analysis today is actually very good at pattern matching and identifying those types of issues. And on a code base our size, you see a lot of them. And static analysis is actually a very important additional process, I'm not going to talk a lot about that, to layer into a robust development process.

All right, so the next piece. All right, if we want to find bugs we want to drive functional quality, we need quality testing. And your game development's messy. You're never going to find all the bugs. And really testing is only good to find bugs. No matter how much testing you do, you're never going to know what bugs you haven't found, obviously. So we've got to make sure we do testing.

And there's a lot of different types of testing. There's the engineer doing their own software testing. There's our QA department who does black box testing. There's automated testing. There's a whole spectrum of testing that we do on our product to help make sure that the product we put out is amazing.

But inevitably, game development's messy. No matter what we do, things do slip out. In fact, in the first 48 hours of Madden launching, there is more testing done on our product by the

gamers than we do an entire 10 month development cycle by ourselves.

So one of the big processes we have in place today is test automation. You can't hire enough QA people to test your game for the scale and the size of games and complexity of games today. So we do lots and lots of test automation. And basically the goal is to get hours and hours and hours of testing through automation on hundreds of dev kits that sit idle at the studio at night when people are at home.

There's one big piece that I want to talk about with test automation, which is when we talk to engineering, verification, and validation, testing is a means to compare the execution, the results of the test against the expected results for that test. And one thing that we're starting to do a lot more at EA is we're kind of looking at testing where automated testing in particular where kind of the only expected result is that the test completes. That test may take five minutes. It may take 50 minutes.

All we're doing is looking, hey, did that test complete. And within that span of say 50 minutes, there's all kinds of other things that went on. And we're not really testing whether all those things between here and here met what our expected results are. So really, this is about measurement and functional quality measurement, which starts with technical KPIs.

Over a 50 minute test, did the game run at frame rate? What are the load times between your different modes? Did they meet what our expected requirements were? Obviously there's crashes, desyncs, and disconnects, which are very abrupt failures for that test where it won't actually complete. But there's all these other technical KPIs where the test will continue, but it may not have been doing what you'd expect it to do.

AUDIENCE: [INAUDIBLE].

TIMOTHY COWAN: So what we're starting to do more now is measure and make sure that every build so that every test where an automated test, no one's sitting there watching it. I may not know. If a QA person doesn't sit there and watch that automated test, we don't know that it's not running at target frame rate unless we measure it and report it.

And with online services today, you can do a lot of measuring and automated reporting, capture all that data, and create dashboards that can show, all right, what's the performance of my Game this is one the first ones we put in over the last couple years. We've always had performance analysis. But what we now have is automated solutions with very detailed level of

reports, so that every time someone plays our game, we're getting frame rate reports out of that game.

And you can drill down a little bit to look at for Madden by stadium. Every game that's been played in each stadium, what does the frame rate look like? And look and see are there certain stadiums that have problems. And you can even look at an individual session and say, all right, was there an individual game session that had a strange frame rate problem?

Now beyond our technical KPIs, those key performance indicators, there's other types of functional issues that we need to look to measure. Bugs do slip out into the wild. This was one that got a lot of publicity this year, the tiny titan. UFC had some well known glitches in its game where a lot of funny animation and physics problems. Inevitably when these types of things happen, if you're familiar with the notion of asserts, sort of non fatal errors or warnings that engineers put into their code to catch non fatal problems and warn them that, hey, you might want to look at this.

I guarantee when both of those problems showed up in a debug build, an assert probably would've yelled at a developer and m, hey, something's wrong. So there's this class of non fatal functional problems that are pretty severe, because if they get out into the wild, this breaks the immersion and ability to experience emotion when you play the game. Or at least in some cases, this one's probably a negative emotion.

We need to start to shift our focus on as you build engineering systems. This is, again, as an engineer, what you bring to the table is this methodology and process of verifying and validating against functional requirements. Make sure as you're building your features, you're thinking about testing and you're thinking about measuring. Is this doing what it's supposed to do?

Because I've got a code base of 10 million lines of code, 200 people contributing to it. There's a lot of complexity. And you never know throughout that software development life cycle what types of problems might show up. So as you're going into your design and requirements, start thinking about what types of failure modes are critical that I should be measuring and reporting through online systems. We're starting to do a lot more of this at our studio.

So in summary, we really talked about two things, code review and testing. And specifically measurement when you're doing that testing. But it's really about, for me, an engineer's responsible for driving the functional quality of a product. We write the code, we write the

specifications. We have to drive the functional quality of game software. And a well executed project has both the technology and processes in place to do that and make sure that you're able to take that big project and get it done by August every single year.

So that's it. I think I maybe talked a little bit longer than I originally expected. But we've got a little bit of time for Q&A. I think I was told you guys take a break at 2:00. But you can go ahead and ask any questions and I'll answer anything I can. When we take the break, if one on one if you want to ask more questions, I'll probably hang around while you're doing play testing and can talk more. Any questions?

AUDIENCE: [INAUDIBLE].

TIMOTHY COWAN: Today it is all C++. 100%. I did a talk yesterday sort of to a general audience about how the industry's evolved. There used to be a lot of assembly language intermixed with C++. We don't do that anymore today on today's consoles.

And there is in data we do do Lua scripting. There is ActionScript. There is shader code and shaders for graphics programming. I bucket those more as data. And they tend to be-- we have a Lua interpreter that's written in C++ the can load Lua data scripts and process it. We have an ActionScript interpreter that loads ActionScript and processes it. So generally when I'm talking the 10 million lines of code, it's the C++ game engine client. But it does interact with other types of code that we consider data today.

AUDIENCE: So I've been in code review situation all the way from [INAUDIBLE] mostly automated through some sort of code tracking system or commit system where someone's about to commit something, [INAUDIBLE], or all the way to the other end where someone just grabs me and makes me look at their screen. How does that work? Can you walk us through how review works?

TIMOTHY COWAN: Sure. The policy is all code gets reviewed. With that kind of policy, there's a lot of flexibility in how the teams actually implement the process. But the tool set we give them, we do have a server. We actually use a software from SmartBear called Collaborator.

And when you're going to submit code to Perforce, we use Perforce for source control, you upload the diffs to Collaborator and assign that to a reviewer to go look at those diffs. And they can mark it up, put little bugs and comments, send it back to you, and you can kind of go back and forth. So it's one option. That's how 90% of the code review gets done.

But the thing I tell the teams is you don't have to do it that way. In fact, having someone come to your desk and talk to you and look at the code over your shoulder can, in a lot of cases, be much higher value. Because that can help set some context on why did you write it this way, and you can have a conversation. So that's an option as well.

And you also have the flexibility, depending on the situation, sometimes code gets reviewed after check in. It doesn't have to get reviewed prior to check in. That's kind of up to the teams, up to the urgency of the check in, the seniority of the review. But the request is that all code gets reviewed. Again, it kind of hit the four bullet points on why.

So I guess basically it depends. It is game development, so we try to be a little informal about it and give you the flexibility to do the review the way you see fit. But provide a lot of tools and process and formality kind of through a prescribed automated system as well. I think there was a question in the back.

AUDIENCE: Yeah. How much additional work is for you to do things like hoarding the same game to Xbox, PS1, [INAUDIBLE].

TIMOTHY COWAN: For today's consoles, Xbox One, PlayStation 4. You're really starting to get to hardware that's converging. It's becoming more PC like. Probably the biggest difference is actually going to be in the operating system. Each of those consoles has an operating system. Xbox One ends up being sort of flavor of Windows. And PlayStation 4 ends up being some variety of Linux.

So you've got this operating system and driver layer that is different. But with the way EA's structured, obviously we're cross platform. Madden, when I showed the charts earlier, actually shipped on seven different platforms that one year for Madden 13. We've always built cross platform support.

I characterize it as about a 70-30 to 80-20 rule, where the majority of your code has no idea what hardware it's running on. And there's just kind of this base layer that interacts with the operating system and the drivers and does what it has to do specifically for the console.

And so it's call it 80-20 in terms of code and 80-20 in terms of effort. So that 20% layer generally gets built by our central technology groups, whether it's the base level rendering layer, the base level file system layer, the base level of memory management layer. At EA we've got central technology where that gets built centrally and all teams use it.

And then the teams take that technology and build their platform agnostic game on top of it where it kind of doesn't care whether it's Xbox One or PlayStation 4. But then the other space that is different is just really, I'd say, performance. Each of the systems has slightly different bottlenecks in terms of what's fast and what's not, whether it's memory bandwidth, shader pipelines. They have slightly different strengths and weaknesses. They're tuned differently.

So you tend to have to look at your content and your data and structure it in a way where it's optimal for the two systems. In the space of just kind of art contact, how many polys, how your textures are structured and bundled, how you write your shaders especially. There's quite a bit of space, quite a bit of, I'd say, platform specific optimization work to make sure we're really taking advantage of the different bottlenecks in the system.

AUDIENCE: Why did you choose to join [INAUDIBLE] in the first place?

TIMOTHY COWAN: To be honest, I had no idea that you could make a career in video games. Like I said in the intro, I wasn't looking to go work on video games, mostly because I didn't know that you could. One of the games I played the most growing up actually was Madden. In fact, really football games. Anywhere from Tecmo Bowl on the original Nintendo Entertainment System to Madden on the Sega Genesis. I played a lot of NFL football games growing up.

It was actually the NFL football games that I played that got me interested in the sport of football versus the other way around where I was a fan of the sport so I played the games. It was actually the games as a junior high, high school student that I really enjoyed playing those football games the most and enjoyed beating my friends at them if I could.

And so when I got contacted by the recruiter where I'm going to go work at Lockheed, hey, come work on the next generation of interactive entertainment, make a football simulator, I was like, well, that would be great. I used to love playing those games and was really excited about coming down and getting to work on some of the games that I loved playing when I was younger. Anything else?

AUDIENCE: Do know of SB Nation's "Breaking Madden" series? And if so, do you have any opinions on that? They deliberately set the game up with extreme characters, like 90% skill everything and 0% skill everything.

TIMOTHY COWAN: I did actually. I think it was probably about two weeks ago I watched that. So I do know what you're talking about. My opinion, I think that's awesome. Madden and EA Sports, at its core, is

all about authenticity and realism. EA Sports, it's in the game. And we're pretty serious about that authenticity, realism, almost to the point that, it's just me talking as a gamer, where we tend to be a little too serious.

And I kind of like that kind of stuff where it's more just having fun with the game, which is what it's all about. And giving, I think, the content creation tools in the hands of the gamers, where you're starting to see a lot more gamers love to create just as much as they love to play. And giving them the flexibility that they can go in do, it's an open sandbox, do whatever you want. We've maybe put a few walls and barriers around. But I would love to be able to put into Madden, hey, let's tweak the gravity. Let's really change up the physics.

Now, the core of madness is about authenticity in realism. So there's certain things that we're not going to deviate from that core. But what little flexibility we can put in where you can break Madden and have a little fun. Because ultimately it's games. You want to have fun and that's what you want to do. Personally I love to see that we've got that support in there. And that's where the tiny titan that slipped through. That wasn't intentional.

I think we later just sort of having fun and laughing at ourselves we went ahead released a ultimate team card pack that allowed you to unlock that as an intentional bug. But the original case, it was actually another one of those we make a change and in our excitement about getting that change out to our gamers, we move fast, maybe don't have all the processes in place that we should've. Someone does some rough testing, looks good, push it out, and something slips through. But it's a big game.

When you start talking about that program testing and verification and validation, there are so many different permutations that have to be covered. So many different modes. It's not infinite, but it's a very large finite number. And it's almost impossible to test everything, which is why we're trying to start to get to a lot more rigor in our testing and how we measure and verify that everything is on track.

Let's go ahead and take a break. And I think, like I said, if you have a question that you just want to ask individually, I'll be here to talk a little bit longer. So thank you.

[APPLAUSE]

PROFESSOR: So the clock says 2:08. Back in this room at 2:18. We're going to give you time to start planning. You're going to run two tests. And then you're going to have a little bit of time to

reflect on your test and just a really quick tell us what you learned today.

OK, it's 2:18 or thereabouts. Schedule for today is you've got about 15 minutes to do planning. That means you should know what observations you're going to make when you're doing these tests.

If you're going to do questions, either creating a quick survey or having questions you're going to ask your testers. Just like we asked for last time. You're going to spend 30 minutes doing a play test. That means you're going to set up at least three computers with three observers to run those tests.

At 3:05 I'm going to say, hey, switch. That means anybody who's doing an observation is switching with the people who are doing testing. Everyone should have a chance to both observe and play at least three or four games. That lasts for another 30 minutes.

At 3:35, you're going to time in teams to complete your focus test reports. And then we'll just ask you a few questions about what you observed, but no formal presentation today. We just want to kind of know what you saw today.

Questions? All right. So I'm going to yell at 2:35 to have your play tests ready to start by 2:35.