SARA VERRILLI: OK. Welcome to class. Some very quick announcements. We had the date wrong on the vision statement. The vision statement you guys worked on on Monday is due today. We had the date wrong in Stellar. So it will not be late as long as it's turned in by tomorrow morning. We have adjusted the date on Stellar so now it shows the real date. But since we made that mistake, it's not late until tomorrow. But we will expect them all in by the end of the day today.

Secondly, the things that are due next Monday, you will be expected to be turning in a product backlog. We're going to work on creating product backlog in class today and then you will have some time to refine and think about it as you're working on the project over the weekend.

One thing you should be working on along the way, remember we require a design change log for this project as well. So you should have already started it because you probably made some design decisions in your meeting on Monday. So don't forget that and don't try to do it at the end. I think that's all the announcements. Yeah? OK.

So today's class is going to be, we're going to start talking about project management. What we are trying to do with project management is give you the just in time version of it. Which is to say, we're trying to give you the pieces of project management you need for the project you're on right now in the state it's on. And we're going to be just in time project management for project two.

And then on project three, we're going to expect you to use all those pieces from the beginning and all the way through on your own so that you'll have had a walkthrough on project two, a practice run on project three, and then on project four hopefully you will need it and it will all work. That's sort of the philosophy.

So we are giving it out to you in bits and pieces, and I acknowledge that. I suspect that between today's lecture and Monday's lecture, we will have covered sort of all the things we expect you to be doing as far as project management. But today's lecture's sort of the first half and then Monday's will be the second half and tidying it all up.

So we're going to do a brief history of how project management was done in the past in the software industry. We're going to talk about agile, which is where many software companies are using now or moving to with various levels of success. Not all of them, unfortunately. Then we're going to start getting into the nitty gritty of the agile management system that our lab is most familiar with, which is Scrum. And we'll start walking through all the aspects of Scrum.

We will concentrate today on product backlogs. We'll take a break from our lecture and you guys will have the opportunity in about a half an hour to make your own project backlogs and then do very quick presentations of sort of, OK, here are the top features that we found in our product backlog when we sat down and actually thought about our game.

Then unfortunately I will get up and start talking again. We'll go over the meetings in Scrum, how they run, and what they are, and why you do them, and their level of importance. And when we hit the end of that, we'll have you run a small one yourself and then we will actually get to use the rest of time in class for working in your groups and working on your projects. OK?

Any questions? And let's go. Now that we have the class interest guides, I can mostly skip that slide. OK.

So you all took this class to learn how to create video games. We, however, created this class to teach project management in video games. Fortunately, we're all going to get what we want, because you really do need some good project management skills to successfully make video games. The better your project management, the better your chances of coming out with a good game. It won't guarantee it.

You need a lot of other good things as well. But I can pretty much guarantee that without any project management, your attempt to make a game is going to fail. So do try to listen. I know this is some of the less interesting stuff we cover, especially coming after the high of prototyping and making games. But this is just as important as prototyping.

So what is project management? Let's start by explaining what the heck I'm talking about. Project management is the work on the project that's devoted to planning, organizing, securing, motivating, and controlling the resources to successfully complete a project. You've all heard of fast, good, cheap, pick two. Most projects are actually lucky if they manage to choose one. And it's project management allows them to figure out whether they're going to

be fast or good or good or cheap or whatever.

Project management is all about predicting. I knew the definition up there says controlling, but you really can't control a whole lot of things. What you can do is you can make some estimates, make some predictions, plan for the risks you're going to run into and then deal with them. So it's really about predicting what you're going to need to do when you're going to need to do it and how you're going to need to do it.

If you follow game and other large software project releases, you know how rarely they actually manage to come out on time with all of their features. Games are especially famous for coming out late, for being over budget, and for often under delivering the shiny things that they had mentioned. Can anybody think of a couple of games recently that have done that? No?

AUDIENCE: [INAUDIBLE].

SARA VERRILLI: Yep. OK definitions may vary. But yeah, it is really, really common for games to come out late and over budget. Here's another one. Anybody name a tripe A game that shipped on time? I'm not actually sure I can do this one. With all the features that they predicted. Can anyone name one?

AUDIENCE: [INAUDIBLE].

SARA VERRILLI: Call Of Duty. OK. That's impressive.

OK, so how do you manage to plan what you're going to need to do, get all of your resources in order, have them all in the right place at the right time, and get them working together? The old model of doing this, which is actually the model that I worked at in several studios and I've seen several other studios that use it. And it doesn't work very well.

But sort of the model that things started with in the industry and grew from is waterfall. And I want to talk about that because it's where some studios still are and it's where a lot of software projects still are. And so it's good to understand what the history of project management is so you understand some of the problems that were found in that model that the agile model is trying to fix. Agile has its own problems and we'll talk about them. But let's start with waterfall.

So the very traditional tool is waterfall, where you prepare for every stage. You sort of finish every stage 100% and then you move on to the next stage to see what happens. First you

need your spec. You need your requirements. Which in game design is sort of your concept and your design phase.

You figure out what you want to make, you figure out what all the features are. You also figure out other constraints like what is your budget, how long you're going to take, what can the people on your team actually do. If you don't have a physics programmer, you probably don't want to make a physics game. All that pulling all those constraints together to make a design that is reasonable that works with the things that you think you're going to have to work with.

That concepting and design is going to flow into pre-production where you sit there and you made a whole lot of assumptions in design and concept. And in pre-production, you test them in very small scale usually. You figure out if people can actually do the things they said they were going to do. You figure out if the software you were going to buy is actually going to work or not and all those other details. Using the data you gathered in pre-production, you create a whole lot of big fancy schedules.

And using those schedules, you hire up your team and you go into production, and you implement everything and you get it all built. Well, you get most of it built. Then in game processes, you go into alpha when you've got most of your features in. And that's when you usually bring on a few small testers to do a little bit of testing and make sure your code is working the way you expect it to.

Then when you've got all your features in, it's mostly working, you go into beta. You bring on more testers. In big studios these days it's often common to have open betas. If you're looking at some sort of massive game where you're not going to hire 100,000 testers. So what you do is you have an open beta and you let people play the game for free in exchange for testing it. That hopefully nails out the rest of your big problems and then you ship it.

Does anyone see the big problem with games in this model? It's a little subtle, so it's easy to miss. It's OK if you don't. Yeah?

AUDIENCE: [INAUDIBLE].

SARA VERRILLI: That is it. Almost all of the testing is here at the end at beta. It's a little late to change your design if you're doing all of your testing at the end. if you're doing all your testing at the end, the team can't respond to it. They can't fix things. And that is where a lot of schedules get stretched out and a lot of budgets get stretched out. Because all of the planning you had goes

out the door somewhere around early alpha and you have to start reworking your whole system.

The problem here is, of course, you can't predict fun. You can't design an engaging experience without interacting with the people who are going to be using it, which means not just the people on your team, not just the small group of pre-prod. I mean, in waterfall they do do testing. In pre-production you test, in design, you test.

But you're almost always testing with a very small group of people. The other designers on your team, the other people in your company. You're not going for any big testing. You're not going for any user testing. You're not finding out how the people who are actually going to use your product are going to react to it. So in many ways it's doomed to fail. There's not a lot you can do about it.

None of these problems are unique to games. It's why big software projects often run over budget and run into a lot of feature creep. But games have the really unique problem that their primary feature, their primary goal, is to be interesting and engaging to use. If you are looking at business software, and it's hard to use and it's annoying, but it does all the calculations you need and it's the one that the tech department bought and deployed, you're going to use it. There's just no getting around it.

But nobody has to play Call of Duty. Nobody has to turn on a particular game on their phone and play it. So if a game fails in that sort of user connection, it has completely failed. There's nothing else you can do. It really kind of comes down to, if your users aren't happy, it's not a good game. Your design changes, your schedule changes, and your project management has now-- all of the time you put in project management has mostly a waste of time.

So what do you do instead? First of all, you think about building design iteration into your project management model. You assume that you're going to be iterating and changing your design all the way around. That is what the agile project management movement I guess is the right way to put it was invented from. There are actually several different types of project management that call themselves agile and there are several different techniques and tricks that call themselves agile.

The one thing that they have in common is they try to follow the agile manifesto, which is individuals interactions over processes and tools. Does anyone want to-- let's just go ahead. Working software over comprehensive documentation. Customer collaboration over contract

negotiation. And responding to change over following a plan.

So for the first one, individuals and interactions, means it's better to get up and talk to someone than to send a whole lot of change requests over. Working software. That's why we ask for paper prototypes and vision statements instead of a big design document. It's why we want to see what it's doing rather than having you tell us what it will do.

You don't really have so much customer collaboration. So you don't even have to worry about signing contracts and getting people to observe milestones. That's less of a problem in a class specifically. And responding to change over following plan means that if you discover what you're doing isn't working, you sit down and you figure out how to fix it rather than continuing to go down the rabbit hole.

PROFESSOR: I just want to state that everything on this are good things to do. Even the things that are sort of low in priority. It's good to have processes and tools. But it's more important that you have good interactions between the individuals in your group. Say if you are saying, no, I can't actually get up and talk to this person because we have a process that we have to follow. Then you're now prioritizing incorrectly [INAUDIBLE].

I guess the [INAUDIBLE] a lot of your classmates are going to be people who are going to play your games. And you [INAUDIBLE] collaboration is important in that sense because you just replace that with work with your players. Work with the people who are going to play your game.

And yes, we did specify this is what we want you to see. This is what we wanted to see in the game that they are going to create. That's my contract. Well, that's less important than actually coming and talking to us about problems that you might be having and [INAUDIBLE]. Don't just use our syllabus as a laundry list of things that you have to do [INAUDIBLE].

SARA VERRILLI: Yeah. Yeah. Thank you. So the model of making a software project. I keep wanting to say game, but this is for any software, actually, agile model. Is that the project start you've got your vision and your idea what you're making.

And then on every iteration, you go through every step, essentially, of the waterfall process. You do a little bit of design, a little bit of detailed design. You do a little bit of implementation of that particular chunk of the design you've just created. Then you spend some time testing it to make sure that, A, it works the way you expect it to, and B, it's achieving the result you want to

get. I.e. the feature does what you expect it to do as far as improving the quality of your project.

Then you spend some time reviewing what your overall project is doing. Is it still going the same direction you thought it would be going at the start of project? Is that the direction you want to be going in? And then you can modify your plan accordingly and you start over with another iteration.

The big savings you have here and the way in which agile management tends to save time over waterfall management in this project is you spend less time designing features that people don't want and more time working on the features that people do want. And that's actually where the big time savings is. Agile can't make you write code faster. What it can do is make sure that the code you're writing is the code you want as opposed to the code that you end up cutting at the end of the project.

Some caveats before we start actually talking about how this applies to you and your project. Agile is good, not perfect. Which means once you've learned the tools and processes for agile development, you will want to keep adjusting and improving them.

You may discover that we're going to give you a set of outlines and a baker's list of things to follow and a recipe to do project management. And you may discover that following some portions of this recipe does not work well for the group you're in. Well the right thing to do is to set those pieces aside and figure out a way to work around it. A good agile team is always adjusting their processes and trying to improve them.

Agile assumes that all of your developers are interchangeable, which they're not. Different programmers have different skills. And on a game project especially, you have people who can do art, people who are better design, you have people who are better at x or y or z. They're not all interchangeable. Which means that it takes more work to set up what your task set that you can get done in any particular period of time on a project than pure agile was. Agile says, OK, you've got this list of tasks. You've got enough time to do them. They'll get done.

When you're making a game, you have to look more closely at the dependencies and who can get each of those tasks done to make sure that you have not in fact arranged for two of your team members to do nothing this sprint and two of your team members to do twice as much as they actually have time for.

Sprint is the iteration cycle. I will explain it again later. I'm trying not to jump ahead and use jargon I haven't introduced, but it's sometimes a challenge. Let me know if I use a phrase that you don't understand, because I will explain it or redefine it for you.

Finally, and this is a problem you're likely to run into a little bit less in class, but you are more likely to run into when you go into the outside world and you work on teams with different people. People trained in different disciplines communicate differently. They have different terminology.

On game teams, this is really hard. Because you will often find, and I'm not actually trying to stereotype all programmers talk like this or all artists talk like that. But it is the case that the people who do programming and the people who are creating your assets, be they art or design or audio, often will talk past each other on specific problems.

The artist will not be able to explain why they can't meet the programmer's technical requirements and the programmers will be unwilling to figure out why it is the artist can't meet those technical requirements. They've given the technical requirements. Why don't you just make something that works in it?

Since agile depends on really good communication and the willingness to solve problems like this, this can create a huge tie up, especially in an agile team. It can pretty much stop anything from getting done. So that's another big red flag to be aware of when you're using these methods.

OK. So there's more than one agile system. The one that we use, that we have the people work in our labs [INAUDIBLE] use, is called Scrum. Is anybody familiar with Scrum? Yay, there's a few people. Anybody used it at internships and things like that? OK. All right. So you guys can help your teams out a little bit hopefully.

So Scrum has three buzz words, and I may as well go ahead and call them buzz words, because that's really what the are, that are supposed to define it and let you understand what it is about. They are transparency, inspection, and adaptability. You can think of them as Scrum's vision statement. Much shorter than a document.

Essentially, if you are doing Scrum right, all of your decisions and processes are transparent. Everyone on the team who needs to be involved in a decision is and anyone on the team who

wants to understand why something was done can. No one on the team should be able to say at any point, well, no one told me about that. Because it's both the team's responsibility to share information and each team member's responsibility to stay informed.

Everything done by the team is inspected at some point. Decisions, processes, team actions, deliverables, code. The team thinks proactively about what they're doing, how they're doing it. And if the team discovers a problem, for example, the build keeps being broken every night, the team doesn't just say, oh, we'll work around it. Instead it takes responsibility figuring out how, why, and fixing not just the broken build, but the processes that led to the broken build.

And finally, which feeds back into the other two, the team is willing to adapt any processes and decisions to the reality they're actually dealing with. When they find a structural problem, for example, how does art assets getting in the game? Well right now we're having the programmers hand load them in and we're losing six hours of programmer time every week.

Instead they sit down and say, well, how do we fix this? Do we take 12 hours next sprint to code up a tool that allows the artist to put it in, and then we'll save all that time going forward? Do we teach the artists how do put their art in directly?

I've had this discussion on a team in which we were losing an awful lot of time to programmers putting art in game. And the programmers were convinced it would be faster to just keep putting it in because it would take too long to teach the artists how to do it, despite the fact that it was eating eight or 12 hours of programmer time every week. I still think they were wrong. Anyway.

Note that I keep saying team and team members. In Scrum, the responsibility for project management doesn't sit with a producer or design lead or the programming lead. It's the team's responsibility, guided by the team members, to manage their own project. That doesn't mean that a Scrum team gets to do whatever they want. They still have constraints, requirements, expectations, project handouts handed down to them from managers, clients, instructors, and other people.

But in exchange for taking responsibility for the project management, Scrum teams do get to explain to the producers, managers, executives, and outside forces just how much they think they can achieve in a given time. So you can go ahead and ask a Scrum team for the sky, the earth, the stars, and the moon, and an experienced and empowered Scrum team will explain that, OK, in that time and on that budget, we can get you a terrarium with a fancy night light.

So how does this work? With vocabulary. There may or may not be a quiz later. So the project cycle. This is the cycle of a whole project with the internal cycle right there in the middle where you can see it. Looking at it from sort of the whole project life, you've got pregame sprint, which is-- let me start with this.

Let me start with what a sprint is. So a sprint is the way that a Scrum project divides up its design iteration cycle. Sprint length depends on the project and what the team wants. One month was an early standard. But a lot of people are now using two weeks instead. In this class, we encourage one week sprints, given the length of the projects and the sort of reality of how much you can or can't get done in a week. So a six month project can be divided into six one [INAUDIBLE] sprints, 12 two week sprints depending on what the team wants to do.

So much like a waterfall project, you've got your design and vision time. You've got your development time and then you've got your wrapper time. Your vision time is going to be when you're coming up with the idea and the overall outline. In many ways, that's what you did on Monday. That was kind is you formed teams, you got a vision statement. Some people were even sitting around and making feature lists. That's really your pregame staging sprint right there.

In the development sprints, that's when you're actually making the game. And you're taking a couple of features every sprint. You're getting them working, you're testing them, and then you're moving on and grabbing a few more features, getting them working, and testing them until it all comes together.

Product backlog. Does anybody here know what a product backlog is? Go ahead and tell me.

AUDIENCE: [INAUDIBLE].

SARA VERRILLI: Exactly. Actually not in a sprint, in the--

AUDIENCE: [INAUDIBLE].

SARA VERRILLI: In the overall project. Yeah, the product backlog is in fact the feature list for the whole thing. Once you've got it, you take bits of it off to make your sprint backlog. The sprint backlog is what gets done in every sprint. Eventually hopefully your product backlog is at 0.

And some teams, I'm afraid of your teams are probably not going to have luxury of this, take

what's called a wrapper sprint or two to kind of hardening. You can think of it as polish and hardening. Taking the time where you're not putting in any new features but you're making sure that all the systems really do work. They're kind of shakedown crews. So that is the overall project plan.

What does it look like on the inside of the spirit? Let me see if I can see what that looks like. There we go. So here we have our big pile of definitions. So since you're doing a little bit of project planning, design implementation and testing in every sprint, you have to figure out when and where you're doing those things. Scrum has a fairly I don't want to say rigid, but they do have a very well defined set of processes and tools to do that with.

So to handle project planning and communication, every sprint involves a set of standardized meetings. Sprint planning, daily Scrum, sprint review, and the retrospective. To handle the features and task management, Scrum teams create and use a set of tools. Artifacts is what it's called. The product backlog, the sprint backlog, the task list, and a Scrum board.

At the start of its development sprint, the team turns to its product backlog, which is, like we said, the feature list. Holds what's called a sprint planning meeting in which they decide just how many of those items that can get done. That list of items that they're moving from their product backlog becomes the sprint backlog. From there they break those features. Because features coming off of a backlog, usually fairly large.

So they'll break them down into much smaller items called tasks. A task is just a chunk of work that you can estimate fairly well. Having gotten that big list of tasks, they start working their way through it. Every day while they're working on the project, you'll hold a daily Scrum meeting where you talk about what you're doing, how you're doing it, what you hope to get done.

At the end of the sprint, you end up having a sprint view where you take a look at what you've got done. You compared what you wanted to get done with what you actually did. You go ahead and you adjust your product backlog if you need to. And then you hold a retrospective meeting.

And that's where instead of talking about the product and specifically you're talking about the team and the processes. That's where agile tries to improve the way the team is working together is the retrospective meeting. And then you're ready for a new sprint planning meeting and the whole thing starts over again.

OK, I'm almost done throwing vocabulary at you. And then I'll go back and make sure that I actually did define all the vocabulary I've used. So on the team, I claimed that the team was responsible for all of its project management. That's not quite true. Teams typically have three kinds of members. There's you're sort of average everyday team member, who's someone on the team working to keep things running.

The team has a product owner, may have a product owner. In classic Scrum, you always have a product owner. Your teams probably won't. It's much harder to have a product owner on a student team. And the reason for that is because the product owner is the person who holds the sole vision of the product.

For classic Scrum teams, the product owner is the person who creates the product backlog, helps them prioritize it, and lets them know which of their features are the most important and whether or not the features as implemented are working the way they need to. Your team may be lucky enough to have someone who has a good enough grasp of the project. And if you do, it's really, really valuable.

It's really helpful to have one person who you can turn to and say, is this going to work or is this working or is this what we had in mind? And have one person answer that instead of having a big debate over whether it does or not. But on student teams, it can be really hard to have someone in that position. So that's your product owner.

The other unique person on your team is a Scrum master, and you will need one of those. The Scrum master is the lucky person who runs the meetings, serves as the facilitator and the mediator, and is in charge of making sure that Scrum is followed and the team is working reasonably smoothly.

They frequently get stuck with updating product backlog, sprint backlogs, task lists, Scrum boards. Whatever record keeping paraphernalia the team feels it needs to have. Setting up meetings and making sure they happen.

Try to think of the Scrum master as a bit of the team coach and as the keeper of documents rather than as the producer or the person in charge of the project. Just trying to keep the backlogs straight, keeping meetings flowing, and keeping communication following is a lot of work for one person. You don't also want to burden that person with being in charge of leading the team and making all the final decisions.

They often end up drifting together. And we see a lot. And we also make a slip of calling the person who's serving is the team Scrum master as the producer. See if you can try to keep those two roles separate. If you can let the Scrum master just be the person keeping the team running and you can let someone else step into that kind of vision holder, guide the team role. OK?

All right, I think that that is, in fact, what I have to say on that. Everybody think they have at least a definition or an idea for all of these words? Here's the quiz later you see.

AUDIENCE: [INAUDIBLE].

SARA VERRILLI: OK. Should I be mean and choose the one that I haven't defined that nobody admitted to? OK, who can tell me what a Scrum board is?

AUDIENCE: [INAUDIBLE].

SARA VERRILLI: That's exactly what it is. So what a Scrum board is is it's frequently, if you've got a team that all works in the same place, it's usually a big physical board that you have room to put cards with all the tasks on it on. And you just have it divided up into sections and show the status of each task.

By using a Scrum board, teams can figure out exactly how far along they are, what the state of their project is by just looking at the board. Teams that are going to end up working in a much more distributed fashion and who don't have a designated place to sort of leave a Scrum board set up.

For example, you guys are not going to be able to Scrum boards up on the back of the classroom and come in and check up on them every couple of days. There are a couple of online options. And I will talk about them when I actually talk about Scrum boards a little bit more, asking you to get one set up.

PROFESSOR: Just out of curiosity, who has already got something like that? Like an online site with all your tasks on it?

AUDIENCE: We're using Trello.

PROFESSOR: Trello uses exactly a Scrum board style visual representation.

AUDIENCE: There's also a Scrum plug in for Trello for Chrome that we can get points if you're [INAUDIBLE].

PROFESSOR: I did not know that. That's new.

SARA VERRILLI: Oh, that's new.

AUDIENCE: It's called Scrum for Trello or something.

PROFESSOR: OK.

PROFESSOR: Scrum for Trello. I'll google that. What else? I thought I saw another hand go up. What are you using?

AUDIENCE: Asana.

PROFESSOR: Asana? OK. [INAUDIBLE]. So is that just [INAUDIBLE] card like representation, or is it more like spreadsheet?

AUDIENCE: It's more like a [INAUDIBLE].

PROFESSOR: And multiplayers can do this?

SARA VERRILLI: Yeah, I haven't used that site.

PROFESSOR: I don't know any alternatives.

SARA VERRILLI: I will actually talk about Scrum boards a little bit more when we actually have gotten around to making our sprint backlogs and task lists and talking about the techniques you used to make them.

OK, so we're finally actually getting to the action part of this story. And I'm lying there. I'm not actually talk about time boxing right now. I went back and updated my lecture without updating my slides. But we are going to talk about user stories.

So let's talk about how you make a product backlog. Because it is not just a feature list. There's actually some very important differences between a feature list and a product backlog that makes a product backlog a project management tool as well as a design tool. So it's an organized, prioritized feature list. So the team always knows what the most important next feature is.

So if I actually sat down and looked at one of the product backlog you've turned in, every feature in your game should be described or mentioned in the product backlog. And each item should have a priority status indicating how important it is, when are you going to do it. Are you going to do it first or you're going do it last? Or is this the one that's going to get cut when you realize you don't have time to finish the project? So how do you decide what goes into it and how do you estimate and how do you prioritize? There's the big question.

So here is a sample project backlog. It's not for a game but it's for some sort of log in page or something. But it serves pretty much the same purpose. This is sort of the minimum you should have in your product backlog. Some people get fancier and include lots of extra information, but that's not necessary. Keeping it simple, especially to start, will make your life a little bit easier. If you discover there's more information you want to track here, go ahead and do that. But start with the basics.

The basics are an ID number. ID number's not its priority. It's actually in the order of which we thought up the features. This is the fifth feature we thought of. IDs come in useful when you start moving tasks off of your backlog and into your sprint backlog and breaking them down into smaller tasks. So you can use your ID number to know, oh yes, this feature is the same as it is on the product backlog and all of these eight tasks that came out of it are all related to each other, because they all still have the ID number that goes back to the original product backlog.

Then you need a description. This backlog is using user stories, which I'll explain in a moment. But they all have sort of the same pattern. As an authorized user, I want to create new account. As a authorized user I want to. As an unauthorized user, I want to. So that. So there's your description of what the feature is.

Then you need an estimation of how big a job this is. For the product backlog, we don't actually recommend trying to do really fine estimation. They're using numbers in this backlog. I should make a new one that uses my preferred system, which is actually just using the word small, medium, large, extra large. And how big is small? I'm not really sure.

When you first create your backlog, you won't be really sure either. But you could probably tell the difference between a small feature and a medium feature and an extra large feature. So when you go through and you write down all your features, you can say what are all the smallest features here and call this small. What's bigger than that? That's probably a medium.

What's the biggest feature I can see here? Make that a large.

When you're making the product backlog, you don't want to spend a lot of time figuring out estimation and trying to figure out exactly how many hours it's going to take and how many subtasks are involved. What you want to do is get a feel for the overall size of the features and a feel for how big they are in relation to each other. Because if you're fairly good at saying this is about the same size feature is this and this is about the same size feature as that.

Once you realize that feature A took you three hours and you have four other small features, it's not that hard to go, oh, so that's a total of about 12 hours between those four features. So that is the format of a product backlog. Are there any questions? No? OK.

PROFESSOR: Just want to point out occasionally you come across a feature that's actually not one feature. It's actually a whole bunch all rolled up in one. Like multiplayer or something like that.

SARA VERRILLI: Networking.

AUDIENCE: It's not a single feature. It's a whole bunch of different things. If you come across features that clearly warp everything else that you've got on your list, you should try to bring those down and do small and large chunks. So you can be like multiplayer [INAUDIBLE], multiplayer matchmaking. All of these things are separate features.

SARA VERRILLI: But I will say that when you're making your first pass at a product backlog, it's OK to stick multiplayer on there as a feature and then right next to it, extra, extra large. And realize you're going to come back and break it down later. When you're making your first product backlog, you don't want to be worrying about, well, what are all the little sub features of multiplayer?

But you do want to go ahead and acknowledge that, oh my gosh, we've just put something ginormous on the list. And using a very rough estimation tool lets you do that and move onto getting all of your features in. And then you can come back and honestly estimate them a lot more finely when you're actually thinking that you might put them in or when you're actually determining what their overall priority is and how badly you want them. Because if you end up deciding you want to drop multiplayer, there's no point having spent a lot of time dividing it up into 12 or 14 different features.

All right, so I mentioned user stories and I did not actually define them very well, and that's because I was going to come along and define them here. So a user story is just a very specific way to describe a feature. It is, I think, unique to Scrum. Although there may be other

agile methodologies that have stolen it as a way of describing a feature. And we're going to ask you use user stories for project two at least. Just to get you familiar with it and understand why we think they're a neat idea. You may end up deciding that you don't like them and that's OK. But we want you to at least experiment with them.

So the goal of the user story is to help you phrase your feature in a clearly testable and understandable manner to someone who did not help you write the document. OK? So a user story follows the formula of as the person who is going to use this feature, which could be the user. It could be the designer if you're creating a level editor. Could be the artist if you're creating something for them. It could be another set of coders if you're creating tools. But the important thing is to recognize who are you making this feature for and who is going to use it.

Then you want to describe what it's going to do. And that description should be something you can test. So that when someone goes ahead and gets this completed feature on the other end, they can go in pretending to be whoever that person is, use that feature, and confirm that it works.

Finally, you want the end third, which is so that-- explain the reason why you want the feature there. What is the purpose of this feature in your game? What is it achieving? Can anybody think about why this is an interesting or useful format for stating a feature in? Can you see any benefits in it? I can go ahead and tell you what we think the benefits are, but I'm wondering if you guys can think of any.

AUDIENCE: It's testable and focused on the player, or other relative entity.

SARA VERRILLI: Yep. That's one. OK. So it's testable, which means that you can confirm it's working. It's focused on the person who's going to use it. We're going back to that sort of user oriented creation of product.

And finally, having the reason there means that if the person who originally asked for the feature did not understand the game well enough and asked for a lousy feature or one that doesn't work well with the game, the person who is going to implement it can come back and say, hey, you asked for a physics engine.

But you want it so the player can drop rocks into a pool and watch the rocks sink. That's the only use of your physics engine. We can make this a lot faster and easier if I write an animation hack that shows this happening. And we can give you some sliders to control how

fast it falls depending on the weight of the rock.

And people will often say they want something that is bigger or smaller or doesn't actually do what they want it to do. It's really common when you're writing features. It happens all the time. And so having that reason there means that the team can go back and do a double check when they actually go to implement it that the thing that's being asked for is something that is really needed.

And so it puts a double check right back in the feature. And it's not as good as going back and talking to the person who wrote the feature, but it's at least a warning signal telling you that you might want to go talk to someone and make sure this is really what they need and really what they want.

So I think I already just went over this. So because user stories have that check in them, you can make sure you're actually doing the work you want to do and not some other work. One of the things that agile is really trying to do is focus on making sure that you're not doing extra work. You're not doing work that doesn't need to be done, that's going to get cut later on.

And so a lot of the tools and a lot of the processes and a lot of the language and a lot of meetings are all about making sure that before you go out and put a lot of time into something, it's really the thing that you want to be doing. And user stories are just one more way to add that check into your process.

I'm going to skip on time boxing right now because I've got a better discussion. It's more relevant to meetings. And I'd rather talk about it in meetings. So we'll skip talking about it there.

OK, that is actually the first two thirds of the lecture for today. And I was going to propose that we take a 10 minute break. And when you come back, it will be time to make a product backlog for your team. I'm going to give you half an hour to do it in.

And that half an hour will also include the time to prepare a two minute presentation on what the features for your product are and what an updated vision statement for your game, since things may have shifted a bit when you added new people. No visual aids. Just come up, talk for up to two minutes. It's OK if you talk for less. The faster we get through presentations, the faster we get to other things. And then we'll move on to talking about meetings.

AUDIENCE: Is there a sample on Stellar, a sample backlog on Stellar?

SARA VERRILLI: There is not the sample backlog on Stellar, I think. We think there is?

AUDIENCE: If there is, we'll put it up by the time we get back from break.

SARA VERRILLI: OK. Yeah, in many ways I actually recommend you go into Google and create a Google spreadsheet. Give it an ID and priority and status and going from there. But yeah. OK.

AUDIENCE: [INAUDIBLE].

SARA VERRILLI: Is it there? Yes, sample product backlog. OK. It is there. Thank you, Rick. I forgot to check on Stellar. All right, that said, it is 1:52. Come back in 10 minutes and start working on your product backlogs. But go stretch your legs a little bit.

OK so it's 2:02. If you are not already sitting in your project groups, you should probably go ahead and get together in your project teams. At 2:32. 2:35, we'll give you some to move around. 2:35 we will go ahead and we'll make a list for signing up and we'll run through the presentations. OK?

If there any questions or you want to know what's going on, if you'd like some advice, we'll be all hanging on down here. So please feel free to come and ask us for any advice or any help or anything you need. OK?

OK. One moment. We don't need that plugged in right now. OK. So let's go ahead. I'm guessing everybody's ready for presentations. OK. Then let's ask Comcastic to come on down.

STUDENT 1: OK. So the goal of Comcastic is to be able to place internet service stations into towns so that you can service neighborhood and get money. And then eventually you want to build your monopoly cable empire to become very wealthy.

And so the core features that are necessary in order to enable us are being able to look at the town map. You need to be able to place pieces. You need to be able to rotate the pieces because the orientation does matter.

You also need to be able to have some kind of [INAUDIBLE], so you want like an end turn button or something that happens when you place a piece. And that will also give you the money results for that turn. You also need to be able to know how much money you have and

be able to place pieces only on legal spots and to be able to start the game, quit the game, things like that. So those are the core features of Comcastic.

SARA VERRILLI: Thank you.

[APPLAUSE]

All right, Blind Alien.

STUDENT 2: Blind Alien's team goal is to create a creepy and enthralling game. In order to do that, we first need to have really basic structure, which is what we're going to focus on this week. We're going to focus on having first a start menu, where we can just say the instructions really fast and input the number of aliens, which is going to be important for testing. Because you need to be able to change things quickly.

And then we're going to have the main screen, a character, character movement, aliens, alien movement, and then this weekend we're going to get up together and work on the interactions of those aliens and the character and then work on shooting. So that's what we're focusing on. Thank you.

[APPLAUSE]

SARA VERRILLI: All right. Plunder Winds.

STUDENT 3: Hi. So for Plunder Winds we're implementing a similar game to our paper prototype. So we're a grid based, or tile based, turn based game. And built around three core mechanics. Our mechanics are a random encounter mechanic where every square possibly has either treasure or pirates.

A exploration mechanic where you revealed the risk level of nearby tiles. And so that can change what pirates will do to you if you actually land on that square, something like that. And then a wind mechanic, which controls which directions you can move.

And so our goals for this week are to try and implement as much as our core mechanics as possible so that we can possibly look into stretch goals for our games, like possibly implementing one or two unique abilities to improve playability over next week.

SARA VERRILLI: Thank you.

[APPLAUSE]

Lost Underground.

STUDENT 4: Lost Underground. Scared for a second. The overall goal is to create a game that people will like. It'll be random and will allow exploration. Our goals for the weekend are to create the basic bare bones you could walk around a map, you can drop bombs. Make the most basic part out and then build up on everything else.

The features that we're thinking of planning. The most important features of the game, which are the ones we'll be working on this weekend, will just be being able to move around the board, a board, a general board, dropping bombs, and possibly the most basic, nice looking art that we can.

[APPLAUSE]

SARA VERRILLI: Beaver Evolution.

PROFESSOR: Just a quick point, the most basic art that you can make is probably not the nicest looking art you can make. So just [INAUDIBLE].

STUDENT 5: All right, so Beaver Evolution is a 2D turn based game where you're trying to progress your beaver colony. And one of our main core mechanics of this game is that you have three different choices you can make every turn in order to progress to one of the many wind conditions we have in the game.

So this is going to be one of our main focus for this weekend. We want to be able to implement a board, want to be able sort of select how to build our dams, and we want to make it really easy to incorporate the three different choices we have every turn. So our main focus will be also on creating a UI, which makes it really to sort of figure out what you can do, what sort of resources you have, like how far is your beaver colony progressing.

And I think our second main mechanic, which is including the randomness factor, such as the disasters and that kind of stuff. That sort of stuff can be added later in the projects. We don't need to focus on that. Right now we just want to focus on being sure we can play through the game, make sure that's not confusing to the player, and that you can do it without sort of adding any sort of the features that require more balance.

[APPLAUSE]

SARA VERRILLI: Magi Dice.

STUDENT 6: Hi, we're Magi Dice. We want to make a fun, engaging puzzle games where the player moves around dice on a board to try and using modular arithmetic create 0's and set of combo chains. So our goal for the first week is, first of all, just set up the board and the dice so that the player can see what they're doing.

And then our immediate following goal is to incentivize players to create combos that multiple moves earn them, like, create 0s in a row. And we want to do this by giving them more points and having a pretty UI so they're incentivized to set up and do fun stuff.

[APPLAUSE]

SARA VERRILLI: Sparkly Redemption.

STUDENT 7: OK, Sparkly Redemption. 2D real time game where you shoot monsters and pick up sparklies. And the important thing is that when you pick up sparklies, you either get a different weapon or the monsters change their behavior. That means the core thing we really need to implement is to have a [INAUDIBLE] you can move around, have monsters with at these two different behaviors, because the swapping is important, and have sparklies that you can pick up.

[INAUDIBLE] needs at least two different attacks. So that's probably what we're going to focus on at this time. I guess it'll also be sort of important to have it be easily visible at a glance the difference what the current monster behavior is, what your current attack is, and I guess how many sparklies you have. And so I guess it's not the most, most critical thing to have this week, but we're probably also going to try and get it so that you can see what's going on and not just have to sort of guess at it.

[APPLAUSE]

SARA VERRILLI: All right. OK. So you did product backlogs. On Monday you did a fair chunk of your planning sprint. You finished up sort of the second half of that today by getting your product backlog pulled together.

To close out today, I'm going to talk about the meetings that are involved in Scrum. We're not actually going to have you hold them all here because A, they wouldn't be very useful. It would take a lot of time. And because we're introducing Scrum alongside the project, the timing doesn't fall well as we introduce things, would be the right way to put it. We are hoping you will use these meetings in project three as you start planning out your project and figuring things out. So I do want to talk about how you run the meetings we talked about that occur within a Scrum cycle.

But first I'm just going to talk about meetings. And meetings as a force for good as opposed to how most of us probably think of them, which is, let's be honest, as a force for evil. As a force for evil, they suck away a lot of time. If you have a six person team and you have a one hour meeting, that's six hours of work gone. So you want to make sure that those six hours of work are actually good and useful hours of work. There are three primary tools to do this that we're going to talk about here. We're going to talk about time boxing, having a clear agenda, and having involved participants.

Has anybody here heard of time boxing and do you know what that concept is? OK. Timeboxing is when you decide before you have the meeting or before you start any particular activity how much time you're going to let it take. You give the task a time budget and you don't exceed that time budget.

For example, sprints are just one big time box. You're defining that. Here's the set of things we're going to get done, we have a week to do them in. When we're done with that week, we're going to stop and see how much we got done, and we're going to move on from there and reevaluate. Meetings can do this too. And all meetings in Scrum should be time boxed. It's one of the classic Scrum rules is that meetings are time boxed. They have a limited amount of time you spend on them.

Time boxing acknowledges that there's sort of a maximum amount of time you can spend on any particular problem before any time after that is wasted. It does mean that you won't always reach the perfect solution. Because if you've given yourself an hour to discuss a problem and you've talked about it for an hour and you haven't quite come to a great solution, you stop talking anyway.

You decide what the most reasonable solution you've talked about is and you go ahead and do that. It's sort of an implementation of a theory that the perfect is the enemy of the good. It

may be that if you went ahead and talked for another hour, you would find the perfect and ideal solution to the problem. But it's also very possible that you're just going to spend another hour talking about it. It may be better and it usually is better to go ahead with the best solution you came up with and start implementing that and see how it works. So that's the idea behind time boxing.

Clear agendas. If you don't know why you're having a meeting, you probably shouldn't be having it. The meetings that Scrum recommends come with a built in agenda for what you want to do. You'll probably have to have other meetings. Because at point you'll have to talk about to figure out how your version control is working or how this feature is working or how the art assets go into the game or what kind of sound assets you're going to have.

Before you go ahead and have that meeting, make sure you know what the goal of it is, what you want to come out of it with, and honestly, leading into the next point, who actually need to be there at the meeting. Not all meetings require the whole team. A lot of meetings are really good as just one on one conversations or just a small group of people, a subgroup of people who are working on a particular task and get together and talk about it without including everyone else.

The other half of involved participants is if you are in a meeting, be in the meeting. Don't be coding in the background. Don't be checking your cell phone and texting something. Don't work on a problem set. Go ahead and be there. Accept that you have dedicated the next 15 or 20 minutes to working with your coworkers to solve this particular problem. Get the problem solved, get the meeting over, and then get back to doing useful things. So those really are the ways to make meetings useful and to make them work for you rather than working against you.

All right, that said I'm now going to go ahead and talk a little bit about each of the meetings. So starting at the sprint planning meeting, which you've actually already mostly done. We will do it in class on-- you've already done a good chunk of that, which is making the product backlog. We're going to do this on Monday. We're actually going to go ahead and take your product backlog and break it down into a sprint task list and start making tasks. So we will have a in class sprint planning meeting for your next sprint next week.

But a sprint planning meaning sets the team's goals for the sprint and has the deliverable, its goal, of a sprint backlog and an estimated task list for the sprint. So you know exactly how

many hours of work you think you are going to get done over the next week. And it is time boxed to one to two hours.

For a team your size, one hour's probably good. For an inexperienced team your size, 1 and a half hours might actually be better. So the more you do this, the faster you get at it because the easier it gets.

At the end of the sprint, on the other side of it, is the sprint review meeting. It has the deliverables of demonstrating your working product, in this case your finished project two. It is a chance for your team to stop and review and evaluate your product and decide if that's actually what you want. If you have clients to go ahead and say yes, that's what we want, keep working in that direction. Or to say, oh my gosh, this is exactly not what we were thinking of. Quick, change everything.

So the other thing you do in the sprint review meeting after you've reviewed the product is change and update your product backlog. Because it may be some of the features that you thought were really important before you went into this sprint have turned out to be less important.

Or it might be that you thought of five new features that you really need to put into the game and that it's worth cutting some of the other features you already had on the list. So the sprint review meeting sets up your sprint planning meeting by getting your product backlog in shape to go and prepare for the next sprint. It also is time boxed to about one hour, again, for a team your size.

The other end of sprint meeting is the retrospective, which is not the same as the sprint review. A lot of teams try to smash a sprint review meeting and a retrospective meaning into the same one. And they have very different purposes. Sprint review meeting is all about reviewing your product.

The retrospective is about reviewing your process. It's about talking about the team. Ideally, it kind of recognizes the things that went right, the things that went wrong, and generates a short list, at least one or two items, of things to do differently next time because they will improve the things that happened.

So it's a forward looking meeting despite its name. The name is retrospective. It's looking back at what happened in the past to see what you can do better in the future. If you do your team

retrospective well, it actually will be a lot of what we're asking for in our postmortem write ups.

Because really that's what we're looking for in the postmortems is understanding what you did, why you did it, how it could be improved. We want to know what sinkholes you fell into and how you climbed out of them and how you would build a bridge over the sinkhole next time so you don't fall into it.

Finally, there's the daily Scrum. And this is the meaning that happens it regularly over the sprint. Every day, or if your team is not meeting every day, every time your team meets. The goal of the daily Scrum is to keep the team accountable to each other and to encourage communication across the team.

If everyone's being honest at the daily scrum, there's really no excuse for anyone to not know the full state of the project, especially when it's combined with a Scrum board with all the tasks up. Daily Scrum are also actually called daily standups, because a lot of teams do them standing up in order to keep them short.

Classically it's time boxed at 15 minutes. For teams your size, five to six minutes is probably actually enough. Because all each person has to do is stand up and answer these three questions. What did you yesterday or since our last meeting? So what have you gotten done? What are you going to do today? What's on your task list? And finally, what is keeping you from getting things done?

What's blocking you? How come you can't get things done? And sometimes the answer is nothing and sometimes the answer is something, in which case it needs to get dealt with.

The place where daily Scrum often go wrong and end up taking longer than their five minutes or 15 minutes is when people start mentioning what they're being blocked by and everyone jumps in to help them solve the problem. That's not actually what you do at a daily Scrum.

You don't solve the problem at the daily Scrum. You mention it, someone makes a note of it so everyone's aware of it, you get the meeting over, and then the people who actually need to be involved to solve the problem talk to solve the problem. Once again, it's invoking that principle of, if you don't need to be at a meeting, don't be at a meeting.

So that is my comment on daily Scrums. And I was going to have you guys try and do one today, but I actually don't think that would be very useful. I think you've already really probably done a lot of that in sitting down and talking about your product backlogs. So I think I would

prefer to go ahead and close out my lecture here and give you the rest of the hour to continue working in your teams on your projects.

PROFESSOR: One thing we can recommend is that since most of your teams seem to be planning on meeting up sometime between now and Monday-- and I certainly encourage that-- plan to do this during that next meeting, if it's on the weekend, if it's on Friday, if it's tomorrow. Whenever that is, do this and ask every single person these three questions.

And be prepared, personally prepared, to answer all of the three questions. The thing that could be blocking you is you've got a horrible head cold. Your computer just fried. [INAUDIBLE]. I have a problem with my web game engine. And there's probably somebody else on the team who can help you figure that out. [INAUDIBLE]. It could be another class that's giving you problems. That's also a perfectly fine answer. You want your team to know what's preventing you from getting stuff done.

SARA VERRILLI: The other thing you should do over this weekend and have chosen by the time you come in on Monday is a Scrum master for your team. And from the instructor's point of view, that's the person who is primarily responsible for uploading documents, vision documents, product backlogs, design change logs, making sure that we have a link to your project when the project is done.

Because we don't need everyone on the team to do the joint team turns. We really only need one person to do it. From the team's point of view, should be the person helping you do Scrum. And that's all I have for today. So do you have anything else, Rick?

RICK EBERHARDT: Deadlines. So turn in your vision documents, the ones you just worked on, that you updated with you new team. Turn them into Stellar by the end of class today please.

SARA VERRILLI: I was telling people since we had the date wrong, in case something went horribly awry, they had until the end of the day today.