MARIO GOODMAN: My name is Mario Goodman and this class is about advanced techniques in managing building data in Revit.

The class is being streamed, just so you're aware of that. And there are a couple different social media streams out there. But I'm not going to be taking any questions during the presentation.

I'm going to be talking today about building data. And the information about a team makeup with a project budget and schedule, and so on, is what we would call project data. I'm not going to be talking about that. By building data I mean the information about rooms and activities that actually make up the building itself. And on some projects we get a lot of this kind of data and it's difficult to manage it accurately. And we also have a need to validate it, to prove to the owner that we actually delivered a building that meets the requirements.

And a typical workflow starts with these items over on the left. Most importantly, we have the requirements that come from the client, but we also have the results of our own interviews and code sources and things like that. We process this data and we have deliverables on the right. Historically we delivered what were essentially reports like drawings and specifications. Increasingly we're being asked to delivery models and database data.

The class today is going to focus on the piece in the middle. And you notice I've referred to it as a database system because it really involves two parts. It includes the BIM, in this case Revit. It also includes a database. And the BIM and the database are going to be kept synchronized with one another.

Now you might question this decision. Some people would say all of the information should be in the BIM. I'm not going to really engage that discussion right now, I'm just going to posit that this is the way we're planning to work.

And what tends to happen in real life as we get this data in an usable form. Surprisingly often it's actually in paper. If we're lucky, we get an Excel file. But it's structured in a way that it's designed to be usable by a client who's trying to understand their building, and that's certainly appropriate. But it's not structured in a way that's very usable in terms of managing data in Revit.

So what we're going to need to do is take that data and put it into a database in a much more structured form. Then we can work between the database and Revit. And finally we can produce reports out of the database that report on the actual characteristics of the building in contrast with the program data.

Now this is a fairly common problem. It's fairly well understood in some sense. And people typically at this point purchase software in order to do this. And I'm going to talk in a few minutes about some of the software options, and there's some very good ones.

But I often also see these projects get into trouble at this point. Something isn't working and the teams respond by saying, well, we need more training in the software. But the idea that I'm going to suggest today is that it's not really a software problem so much, it's really more of a data problem.

And the reason that people get into trouble if that they don't really understand their data. So what we really want to do is think about our data and how it's structured, and define objectives for how we want to use it, and then we can implement software to meet those objectives.

I'm going to start by talking about some general principles of objects in databases. When we talk about a model in the sense of building information model, what we're really talking about is an object model. And this draws on some theory of object oriented programming, which comes out of computer science starting in the 1960s. On the one hand we intuitively understand what objects are. They're just things like walls and doors and stuff, and we don't really need to be precise about them. But I'm going to ask you to think about this a little in more computer science terms.

In particular objects have two characteristics, they have methods and they have properties. And methods are things that they do. So a door knows how to insert itself into a wall, that's a method. We're actually not going to be talking about methods very much, we're going to be talking a lot about properties because that's where objects store their data.

Now when computer programmers start out to create a program like Revit, they don't just start making any old objects they feel like. They actually want to structure them in a way that's as simple and concise as possible. So they start out at the top of this diagram with a sort of great grandfather object, and they derive objects from that. And they go through a series of generations where they derive a child from a parent object, and then they derive a child from that object. And we go through a process of inheritance.

And the thing about inheritance is that at each generation in the tree, the objects inherit all of the properties from their parent and then they add additional properties. So when we're talking about data and the properties of objects, very high up in this tree Revit objects pick up something called an element ID. So you can count on every object in Revit having an element ID.

As we start getting down into more specifics like family templates, they start to pick up characteristics like their hosting behavior or something like that. And then when we get down into the families themselves, they pick up very basic values like their omni class value.

And if we keep working our way down this tree we keep accumulating more and more kinds of data. We're going to particularly look at the last three levels of the tree here where we have families, for example, tables. We have family types which might have a parameter to indicate different sizes of the table. And then we have family instances where the objects are specific placements of one instance at a time in the modeling environment and we might, for example, apply the material at the instance level.

Now when we think of properties in Revit, we tend to think mostly in terms of parameters. The term property is actually a little bit more general than parameter, but for our purposes we can consider the terms to be pretty much equivalent. So I want to talk about parameters a little bit.

And there's a couple things that we're not going to worry about. One of them is the distinction between shared parameters and not shared parameters. And the other one is project parameters and family parameters. Those are important topics if you want to get into it about families, they just don't really affect the data very much in terms of the way we're going to be talking about data.

There is one thing I do want to talk about, though, and that is that term family parameter because it causes some trouble. So just told the thought family parameter for a second. And I want to talk about how parameters are aligned with their position in the inheritance tree of the objects.

So if we look at the family, the table family, and then a child object of that are the family types, and then the child of that are the actual family instances, it makes pretty good sense to talk about type parameters and instance parameters for the bottom two levels. But if we try to talk about family parameters for the family level, we get into confusion with that previous use of the

word family parameter.

And it's especially confusing because if you go over into the family editors I've brought up on the screen here, and you go down into the dialogue box there, it actually says family parameters down there. And I think this dialog box was probably written by some computer programmer, he used the right term, but it's really misleading.

So it's typical in the community to talk about these family levels parameters as built-in parameters or something like that. And there aren't a whole lot of them and we don't actually have much control over them. Sort of the main I think you will end up paying attention to is the omni class value. But we really can't add these parameters. Most of our attention is going to be devoted to the types level and the instance level.

Now when you get to the types level it makes perfect sense to go into the family editor and work on the types. Because that's what the family editor really deals with is family types. And so we can create parameters there and set their values and things like that. It gets a little bit confusing because you can be in the modeling environment and edit the type values. And obviously Revit provides this to you as a convenience and there's nothing wrong with that. But when you're trying to understand your data and understand the structure of your data, that can be a little bit misleading.

So remember that the modeling environment is really dealing with the instances and if you're editing types from the modeling environment, you're going into sort of a temporary family editor while you're doing it. And in a similar way, when we're working at the instance level, we're working in the modeling environment. And again it makes perfect sense on the right of the screen here to edit the parameters, though it's kind of interesting that they're called properties in that dialogue box, but in any case it makes logical sense to edit them there.

What's a little strange is that you can go into the family editor and edit instance values. And if you think about it, well what does it mean to edit an instance value at the type level in the family editor? Well it turns out what you're really doing is editing a default value for the instance value.

So again it's sort of a nice thing that you can do in Revit, a little confusing in terms of understanding the data structure. When you begin to use parameters, you often have to face a choice as to how you have some piece of data that you're going to manage and you're wondering how should you implement it in families.

So for example if we have a door and we want to show whether it's red or blue, we could take the option which is the upper line here, where we create an instance parameter called door color and we set its value to red or blue. And then we can use that value in a schedule.

Alternatively, we could create two door types, where a type a is inherently red and a type b is inherently blue. And then by specifying the type of the door we're implicitly specifying the color. There's a third option where we use an instance parameter, but rather than setting the instance parameter directly we use a schedule key to control the instance parameter. So we set something about the door, some kind of style setting or something. It implicitly sets the instance value to the color and we get the same result.

Now the reason I'm mentioning all this is that I think people tend to get a little distracted by the kind of moral argument about which is the right one or not. All three routes are valid. You would tend to use types for things that are sort of fixed and widely used. You might use instances for things that are more variable. You would use schedule keys for things where you want to kind of manage the data a little bit more directly. But they're all valid approaches.

But the point that I really want to make is that they have the same information in them. So when we're talking about the information content of the model, it's exactly the same. So when we get down to that schedule on the lower right there we're going to get exactly the same result.

I want to switch gears here and talk a little bit about relational databases. I happen to be a big fan of databases, so I'm going to preach a little bit here. It's sort of amazing to me how little knowledge there is out in my community of the AEC world about databases. Probably most of you have Access installed on your computer and never use it and your office spends a tremendous amount of money training you in Revit. You ought to change that. You ought to learn to use databases, they're really wonderful things. And part of the reason I'm such a huge fan of them is that they're really powerful, but at the same time they're really, really simple.

So I'm going to give you the world's shortest class in databases here. All they really have are tables and relationships. Now a table is exactly what you would think it is, it's a grid with rows and columns. Now if you're old, like me, you sort of remember file cabinets with pieces of paper in them. And you tended to talk about records and fields on pieces of paper, and that terminology has come through. Computer scientists like to talk about rows and columns, but columns and fields are entirely equivalent in rows and records are entirely equivalent.

Now a table in a database needs to have a primary key. And basically this is one column that we identify as the key column. And it's a primary key because it has two characteristics. One is that it has to be there, it can't be blank. And the other thing is that it has to be unique. So in this rooms table, if we're using a room ID to identify the room, we can't have two rooms with the same ID. Again you don't have to be a computer scientist to know this stuff, you just have to think logically and clearly that has to be the case.

Now when we start talking about the relationships between tables, we start talking about foreign keys. This is a term that sometimes gets people a little bit, but think carefully about the way I'm going to define it. A foreign key field in the table is a field that points to the primary key of another table. So in my room's tables here I have a column called floor which can have any number of values in it, but the values point to a separate table, a floors table. And remember that it points to the primary key of the other table. So the values that we're using are the primary key values in the foreign table.

And this means that what we're in effect establishing is what's called a one to many relationship. And the reason we use that term is if you look at the floors table, since you're talking about the primary key, we can only use values there one time. When we're talking about the rooms table we can use a value like floor one as many times as we want. So on the one side we have the little one, and on the many side we have that little infinity sign.

Now the databases enforce something called referential integrity, which is a sort of fancy term for something that basically means you can't cheat. And when you create, or when you use a value in a table, like for example, if I try to put this value, floor four, into this rooms table here, it won't allow me to do that because there's no corresponding value over in the floors table.

And this is something that is sort of a pain in the neck if you're used to using Excel and you start working the databases, you kind of think, well, why is bugging me about these things but that's actually the real strength of databases is that they enforce these kinds of relationships so that you can rely on them later when you're working with the data.

Now when you work with data, you're going to want to start doing something called normalizing it. And the typical example of this is when you start making up a project list. You start listing the people, and their office, and then you put down the address of their office, and then you list the next person and they happen to be in the same office so you write the

address over again.

And you start saying to yourself, well I'm writing redundant data here. I'm writing the address over and over again. And this is problematic for a couple reasons. One is it's a lot of extra work. But also if those two addresses are different then you don't really know which one is the right one. So the obvious thing to do is to split it out into two tables, and this is a nice example of a related table with a one to many relationship. So the office is the one side, and the many is the team list on the other. So that you only have to edit that address one time.

Now generally speaking, you're going to want to normalize your data. And not only does it sort of make it behave a lot better, but it really helps you to understand your data when you start getting into complicated things like equipment requirements or something. If you understand the logic of why a particular room has a particular requirement because the data's been normalized, then you're much further down the road in terms of understanding what you need to do than if you were working with that unnormalized data.

But it might surprise you to hear that I'm going to also tell you that you can do the opposite, which is called flattening data. And basically this means where you take perfectly well normalized data, and unnormalize it by pushing it all back into one table. And there's some good reasons for doing this particularly when you're working with Revit, because Revit is kind of inherently flat. It is sort of difficult to get at data that hasn't been flattened.

But you really need to be careful about one thing, and this is probably one of the main problems that I see when I work with clients who are having trouble with their data. And that is that they've gotten data from their clients, the building owners, who probably had perfectly good, normalized data to begin with but somewhere in the process of delivering it to the architect it all got flattened out. And the problem is, once it's all flattened out, you don't understand the underlying logic.

So if you have a bunch of requirements for equipment in a room, for example, and you have a long list of rooms, the flattened data doesn't give you any indication of what are the different room types so that you can think of them in groups, rather than having to think of them all individually. So the moral of that is if you're getting flattened data, try to figure out some way to go and get the source data or do some other process to renormalize it. I accidentally went too far there.

We've talked now about objects in an object model, and we've talked about data in a relational

database. And as I said in the beginning, I'm assuming that we're working in an environment where we're keeping the database and the BIM model synchronized with one another. So I want to talk a little bit about what it means to link Revit to a database.

There are different ways that you can approach this. I like to think of it as the categories in Revit correspond to tables in the database. So in that left hand diagram there, if we have area objects in our model then we have an area table in the database. And likewise room objects go to a room table. And so when we talk about a link, it's a correspondence between that group of objects and that table.

Now in that left hand diagram, we're really talking about instances of the objects, instances of areas and rooms and so on. We can also talk about types of areas in rooms. And likewise we can have type tables in the database. The diagram on the right which looks at types, introduces some complexity though. And the problem with the diagram on the right is that if you start with a piece of furniture and you go clockwise around the diagram to find out something about its type, you might get one answer. And if you went counterclockwise around the diagram to find out something you might get a different answer. And in data we don't like it when you can do different things and get different answers.

So what I want to say here is something that I'm going to sort of say carefully so I don't have to repeat it too often, and that is that there are a lot of potential logical problems when you work with data. This doesn't mean that you shouldn't try to do things. There are good reasons for doing all of these things. The issue is that you have to own the problem.

In other words, it's up to you or some software that you're using to have some sort of a procedure or something like that that understands what the potential pitfalls are and prevents them. Anyway, so I'm just going to make that warning once and then we'll sort of note it a couple of other times.

So when we start looking at the actual link between Revit and the database, we start looking down at the parameter level and the field level in the database. And what I'm going to suggest here is that we're going to have a correspondence between parameters in the Revit family objects, and fields in the database. And in particular, we're going to pick one of the parameters in Revit and call it the key parameter. And it's going to correspond to that key field and we had when I was talking about databases.

And it has all of those same issues they key fields have. It has to be present, and it has to be

unique. Now this is what establishes the link between the Revit objects and the table in the database. And this is really a profound concept, because a lot of people when you talk about linking Revit to a database, they think well that's fine where is the link? And they kind of think that the link is sort of some thing that they can sort of get their hands on. But there really is no such thing as a link.

The link is defined by a convention, or at logical relationship. So as long as that parameter has the same value as the field in the database, and we all agree that those two things are going to correspond to one another, then the link is maintained.

But the opposite is true, too. If you go in and you change one of those values then you completely break the link. There's no other way to maintain that link. So you have to be very careful with your key links, because that's what maintains the relationship. And if we took the database away and brought in a different one, as long as the link was valid, then we'd be fine. If the link didn't match it would be completely broken.

Now the other parameters also correspond to fields of the table but there are a lot less sensitive. It's perfectly reasonable for them to be blank. In fact it even kind makes sense for them to be different, so if you have different values for these parameters one, two, three, in the model then you have in the database, you're probably going to want to synchronize them at some point. But there's nothing sort of inherently bad about that it just means that they're out of sync with one another.

Now all of these issues get even messier when we start talking about types. I don't really want to take the time to get too deep in the weeds on some of these things, but basically with types you have to kind of decide whether you're really going to control your types from the Revit side. Which is kind of the way we're doing it in the left diagram. Or whether you're going to control them on the database side. Which is the way we're doing it on the right hand diagram.

So now we have objects, and we have a database, and we've defined a logical relationship between them on a kind of theoretical level. We're going to move now to doing something with software. So we're going to use some combination of software and files and things like that in what I'm going to call a system architecture. And there are kind of three strategies that we might use for this and I've ordered them from kind of simplest to most collaborative.

The first one is this single user mode. So in this diagram on the right you can see we have

typical example of Revit with its local Revit model. And then on the left we have that sort of cylinder shaped thing which represents the database. We might have some other data-like images. And potentially in the middle there's a kind of optional part where we might have some sort of a program that runs in Windows that relates the two. But it might be that we don't even really need the piece in the middle.

The point is that a single user using Revit is also controlling their own data. Now this is very simple, and because it's simple it can be very powerful. You can have a lot of really good functionality here. And I want to warn you not to dismiss this too easily.

I hear a lot of discussion about collaborative work flows. And I think people often sort of assume that if you're going to have a collaborative workflow, you're going to have everybody working live in the data all the time. And in my experience, the most successful projects like this are the ones in which a single individual, or a very small number of individuals, actually maintain the data.

And the one thing that they do that's really, really important is that they keep the data really, really accurate. The collaboration occurs because they publish accurate data when it's needed and in a way that's really usable. So a lot of collaboration can occur around a single user model like this.

Historically when we want to do a kind of more multi-user environment, we tended to do that inside the firewall working on our local area networks or wide area networks. And we're pretty comfortable with this in Revit because we're used to doing Revit work sharing. Where we have a shared Revit model of on a server somewhere. To extend this model we sort of do the same thing with our data. So our database moves to the server and possibly other files move to the server.

Want to keep in mind though that Revit knows how to handle this. It's very particular about having you synchronize to Central and so on to keep the data from getting corrupted. When you're working with data in this shared environment it's more collaborative because different users can get in and edit it. But they can also mess it up so, again this is another example of what I was talking about where it's on you to figure out some sort of a workflow to make sure that different people aren't changing the data in ways that's incompatible.

Now the third model is what we might call the most modern model. Certainly the direction the industry's heading which uses the internet. I find the terminology sort of difficult here, in fact

even the distinction between the networking and the internet models is a little bit difficult. If there are any computer scientists in the audience they'll probably scoff at some of what I'm saying.

But I think from a sort of marketing standpoint, we can say that certain products are really internet based. The general idea is that your data is out on a host somewhere and it could likely be somewhere far away. So you have data information there, you have model information there, and you're accessing it somehow through the internet. Now in some cases you're accessing it through a regular browser, like Internet Explorer, Chrome, or something, but in many cases you're accessing it through what's called a rich application.

And that's an application that's sort of split. Part of it runs locally on your computer, part of it runs on the host. This sort of dashed, curvy line in the diagram indicates that those two applications are tightly coupled. So they kind of behave as if you have a single user application that's running on that remote host. And of course this is very collaborative, because all anybody needs to participate is access to the internet. They could be anywhere in the world, and assuming they have the rights they can participate in the project.

If I put all these together, we see a kind of domain of possibilities for system architectures for doing this. And I don't think you'd ever want to do all of these things at once, but the reason I wanted to look at it this way is that I'm going to talk about some software products. And I want to talk about them in terms of how they fit into this picture.

I want to be clear that I'm showing you some different commercial products. I just grabbed some information off the web and I've provided you with the web link. So I really encourage you to go and talk to the vendors about their products. I'm not comparing these products in a performance sense. This isn't a shoot-out or something.

They're all good products. And they're all very different products, so you wouldn't pick one for being better or worse. You would pick one because it's more appropriate to what you're doing. And again, I've ordered them from kind of the simplest to the most collaborative.

The first one I want to look at is something called BIMLink from IBA. And this tool essentially just links Revit to an Excel file. And for purposes of this discussion we're going to say that Excel is a database. This is a very simple tool, it's relatively inexpensive, it's very easy to deploy and teach and so on.

In terms of the system architecture it's fundamentally a kind of single user model at the bottom here. So you have Excel as your database and its linked to Revit. There's some potential for a shared model by moving both the Revit model and the Excel file to a shared server. But it's not really a multi-user mode. That's really just kind of a multi people can get to it mode. You probably want to think of this as a single user architecture.

The second one I want to look at is some tools is called WhiteFeet. And full disclosure, these are tools that I've developed. It's a little presumptuous to have them in here with the big boy commercial applications, but I did want to include the because I'm going to be using these tools for some of the examples later on in the program.

The architecture here uses an access database in the single user mode at the bottom. It also has some options for creating views and things that we'll see in room data sheets. It's possible to use this in a multi-user mode because it can also work with either SQL Server or MySQL, which are more collaborative type databases. So you can start out a project in a single user mode with Access and then you could migrate the data to a multi-user mode.

Another product that I think that people are getting very familiar with is Affinity from Trelligence. In many respects, this is probably the product that's most appealing to people when they kind of come into this problem. It's designed to do very specifically the kinds of things we're talking about. It's kind of a standalone product, in the sense that it's not specifically designed to work with Revit. It works with a variety of different CAD and BIM products.

So in terms of its architecture when we look at it, it actually has its own version of a database. And it has its own graphics, so there is some integration with Revit. But it's strength is that it's kind of stand alone, so it's not really dependent on Revit. But at the same time that's part of the complexity of it is that you end up with graphics in both its environment and in the Revit environment. It has, to some degree, a multi-user mode. It has some additional features that are designed for a multi-user activity. Again it's not so much a multi-user accesses, it's a kind of way of handling multiple projects simultaneously.

As we get into the more truly multi-user products, one that's been around for a long time is CodeBook. This originally came out of Europe in the health care industry but it's quite widely used now around the world and for a variety of project types. This is a very mature product in terms of functionality. It's probably one of the better ones. In a system architecture it was

actually originally written to use an Access database. So it can function very well in that single user mode using simply Access. It more recently also has an option to use a SQL Server database, which is the way you would use it if the data was on a local area network and using client server technology to access the data.

Now one of the things that you might have expected me to be talking about with these products is how their data is actually structured. And I'm not because it's sort of too complicated and I don't fully understand it. But for me, as I get into these products one of the things I'm doing is sort of trying to reverse engineer how they're looking at their data. And I mention that here because I think if there's any product that you would be sort of likely to go into the back end of the database and work directly with the data, it's probably CodeBook. And people often do that, particularly when they're using the Access version.

And then when we get into the more internet based products, one that is certainly getting a lot of attention lately is dRofus from Nosyko. This is again another product that has come from Europe in the health care industry and is now being widely applied around the world for many different project types. And this is one that is built from scratch to be an internet product. So it follows the model we talked about with the internet, it has a pulse grey database. It also is particularly focused on using IFC models. So it's designed to work in an environment where other people are producing IFC models. So it actually can use IFC graphics along with the way that it's working with Revit.

And of course it's very collaborative because you can have users coming in through the, they make an internet connection and use a kind of rich application. The collaboration comes at a bit of a cost with these things, because there's some difference between what's going on with the IFC model and what's going on with your Revit model. So you have to be conscious of that.

And then the last one I'm going to look at is Onuma Planning System. And this one's fairly hard to describe. Onuma provides a lot of different kinds of services and middleware, and I'm not sure there's any sort of one product that I would say is Onuma. The thing that sort of strikes me about what he does though, is that it's very publicly oriented. So he has an internet model that is a little more browser based than the dRofus model. So it's really appropriate when you have situations like with universities or communities where not only is there a big planning team, but you want members of the public to be able to participate a lot. So all they need is a browser to come into these.

Anyway as I said, those are all valid options and I encourage you to talk to the vendors to get probably much better description of what they do than the way I described them.

I want to change gears here now a little bit and talk about some of the workflows that would go along with this. As I said I'm going to do this based on some of the White Feet tools that I've developed. But I think you could sort of imagine doing similar workflows with the other products.

And I'm going to do try to do a little bit of this live here. So to begin with I've got a session of Access running here. And this is the database and over on the left here you can see the tables in the database. So for example, if I look at the program elements that are in this table called program space. And this is just what a database table looks like. We'll come back and talk about this table a little bit more.

But what might surprise you is that there's some other tables in here that we would call system tables. And for example this one that lists the parameters in Revit and the tables and the field names in the database and shows that relationship. So when I was talking earlier about how we have these conventions of which parameters in the Revit model correspond to which fields in the database, you probably were thinking well that's fine but how does how does the system know which one hooks up to which one?

And the way we manage it is we store this information, which we would typically call the schema, we actually store it as data in the database. So you really need to know is the name of the database. And once you've got into the database then you can find out all the other information about the linking that you want to do.

And the other thing that I've loaded for this example here is a Revit model. At various times in this work I'm going to be linking Revit to the database. And the way I'm going to do that is to go to these tools here, then I'm going to pick this Revit database link. And it's going to read the database, and you can see that it's showing you that there are two link types here. I've linked the areas and I've linked the rooms.

And then what I would do is I would query the model. And essentially all that did is go through the model and find all of the areas. So in the top window here we can see the data as it comes from the database. In the bottom window we can see all of the areas that the program found in Revit. And then if I pick this command to compare them and then this command to

synchronize them, all it's really doing is showing me the difference between what's in the database and what's in the Revit model.

And the tool is designed to be sort of un-intrusive in that sense. So there options here for or how to respond to this, you could have one of them govern the other end and fix things. But the point is that the tool is not telling you what to do. You have to understand the database relationship and know what it is you want to do. What the tool is really good at is telling you when the things are out of sync and allowing you to fix them in some way.

So I'm going to get back now to my PowerPoint. And the first thing that we need to do is what I started to talk about earlier, which is to formalize the space program. Now if you look at the far left image, the data as we get it from a client is typically structured in this way that is kind of readable but not terribly usable.

What we want to do is get to the second table which is the same data structured in a proper database format but expressing the space program requirements. From that we're going to derive the third image, which is a table of rooms in which we actually have a list of specific rooms. And then finally, in the fourth image, we can now list, we can now have those rooms correspond to rooms actually in Revit.

And as we do this we're going to want to keep track of certain characteristics of the rooms. And each product here, each project is liable to have different ways of doing this. But I found that looking at two aspects of the problem is typically very useful. One of them is what's the organizational use of the room. In other words, what's its department, our who uses it, or something like that. The other one is its room type. So these are the physical characteristics of the room as to whether it's an office, or a laboratory, or something like that.

And when we define the space program, we look at the program requirements in terms of those characteristics. So we say for a particular organization the type combination we have a particular quantity and an area required for each one. And before we really get into designing a building though, we have to recognize that program requirements at that level are too granular, really, for architecture.

So I'm going to talk a little bit about how I, as an architect, would approach designing a building. And one of the first things I think we would want to do is to come up with some broader areas which are the fewer in number and include an aggregation of rooms. I've called them aggregate planning areas. And the point that I want to stress here is that we're going to

create these areas out of the room requirements by combining the requirements in some way.

And the important idea is that you always want to start out with the most granular definition of categories and types. And then you can combine them into something that's useful for doing the planning. What you don't want to do is try to do the opposite. If you try to work with the aggregate areas and then figure out later on how to subdivide them, that's always much more difficult. So in any kind of data it's always easier to combine granular data into groups than it is to figure out some way to divide something that's been pre-grouped into pieces.

I also want to bring us back to reality a little bit. When you're in the software business you sort of get carried away about architecture. And real architecture is governed by things. We sometimes call this the Parti.

So in this example it's a classroom building. It has a site. We've decided to use that sort of U shape. In the middle image we have some typical dimensions for a classroom and a corridor. In the right hand image, just with some simple math, we can say well we need to have three floors. We know something about the construction type, so we know the floor-to-floor height. We have a lot of sort of context for what we're going to be doing.

So the first thing that we're going to want to do is, using that linking process that I described earlier, we're going to want to get these aggregate areas out of the database and into our model. I didn't tell you how I created the aggregate areas. That's just too much detail, but the point is that I use that linking tool to bring them in as areas. And these are some examples from real projects. And in the upper left there we did them as strips of a fixed height because these were laboratory planners and that was kind of the way they like to do their bubble diagram.

We also have tools that we can then convert those areas from being flat areas into being 3D solids or masses. And then you can begin to use these things, which are of course colored by their organizational value, or their room type or something. And you could begin to shuffle them around and to develop the building massing.

But just continuing with the simple example that I did earlier, you can see I brought in these major areas and then on the right hand side here I've used area boundary lines to begin to define the building. And then just drag those areas in there to lay out the basic plan. And then, doing this for the three levels, then I get the basic form of the building and the basic arrangement of the spaces.

And then even at this point we can begin to validate. Because we've been systematic about our tracking of these areas, we can compare the areas in the model with what was actually called for in the program. And you know what's going to happen here, the model's going to be bigger than the program. And you might as well deal with it at this point, because that's going to become your cost overrun later on. The problem people usually have is they don't know it at this point. They don't figure that out until much further down the road.

Now we're ready to start making actual rooms. And one of the things that you want to do as you go down this process is develop short key values for these uses. And the reason I do that is that you can put those key values in labels on the drawing, and then you keep the longer name for the Access reports and things like that. But the important thing is that as we move from the space program table on the left to the rooms table on the right there's a basic difference between them.

In the space program for an organization and type combination, we have a count. So we're saying for the education group conference room, we need three of them or something. When you get to the rooms list on the right you don't have a count anymore. You have a separate record for each room. So if there's a count of two on the left, you're going to get two records over on the right.

But as we make this transition, both of these tables relate to the organizational uses and the room types with the foreign key relationship. So we're maintaining the integrity of these room characteristics as we go through this process. Now the were the way I built these rooms from here was to use a little VBA program. And this is sort of another opportunity for me to preach here a little bit, because it is the 21st century.

Not that everybody has to write code, but somebody in your office needs to know how to write a simple VBA program. Because if you don't have that skill, you're either going to do a whole bunch of manual work, which is tedious and error prone and then your project manager's going to say he wanted it done differently and then you're going to have to do it all over again.

Or you're going to have to do this, you're going have to buy some product it tells you explicitly how to do it. And you're not going to be able to do it the way you want. So I'm not going to explain the program but it's actually pretty simple. The thing by writing your own program, you can really care about how exactly do I want to number the rooms? And that that's very

important to each particular project.

So anyway what we want to do is create the rooms now. So what I'm going to do is switch back over to my, if I can remember what I did with it, this is my Access here. And I actually think I know what I want to do. I want to go sort of zoom out here and then I'm going to come in here and I'm going to use the room tools. I'm going to place the un-placed rooms. And you can see there's a whole bunch of rooms here. And I'm just going to select them all and create them. So it's creating 31 rooms here. I'll close that.

So you can see it placed those rooms and it sized them according to their required area. And it's also bringing along their other characteristics. So then what I can do is I can grab these rooms and drag them down into where I started the architecture here. And typically you would move the tag at the same time. And so now I'm actually creating rooms that not only have all the right names and everything, but they have the same characteristics in terms of their organizational type and room type characteristics.

Anyway so I'm going to close this now because I don't think I want that one anymore. And to switch back to PowerPoint here. So this is the same model, just a little further along. So I've done all of the rooms and I've numbered the rooms. The next thing that we want to do is validate the program with the database. And this is similar to what I was talking about for the planning areas, but now we're going to actually do it room by room.

So if I switch over to the Access database, if I could remember what I did with it here, here it is. We have some reports. And so for example this report compares the programmed area to the actual area. And it sorts first by the organizational value. So it's saying this department, 100 facilities, has all of these different room types. And in the first two columns, it says how many of them were asked for and what area was asked for. In the second two columns what did we actually get in the Revit model and what's the area. And then in the final two columns what's the difference.

And then because of the way we've structured the data, we can easily sort that the other way. So we could say for these particular room types, these are the departments that have asked for it. And again, what was asked for and what was provided and what's different. And those are the kind of reports that really give you power in a meeting because you really know what's going on. And it's not that you've done anything wrong, it's just that you've had to make certain kinds of decisions in order to make the building work. And if you know what you're doing you

can defend those decisions.

I just want to mention in passing that there's some other uses for the database. Once you have this database going, you can use it to do doors and hardware. You likely will want to do some things with furniture and equipment and so on. Just for the sake of time, I'm not going to get into any of those things.

I do want to spend a little bit of time talking about room data sheets. So we typically create these on large projects. As we start the room data sheet process, one of the things we need to do is number the rooms. And you might say, well I already numbered them when I was working in the database. But what I did is I gave them a room ID in the database. And that's an identifier that relates the room back to the program requirement.

And that's really different from the room number, which is something that appears in construction documents and helps tradespeople to find the room. So the room number is typically numbered sequentially down a corridor, whereas the room ID value is somehow or other based on the program. And I want to encourage you to sort of resist the pressure that you'll get from other people to make those the same number. Because that just leads to a lot of frustration.

Now there are a couple of conditions where we would build room data sheets. It's typically done during the sort of early design, or programming stage, looking at your typical rooms. Or it can be done later on in design development looking at actual individual rooms with more detail. The process is similar in both cases. We also have two workflows that I'm going to show. One of them is working entirely within Revit and using views, the other one is working through the Access database.

So if we're working in Revit, what we want to do is make a little plan for each room. Then what we're going to do is tag the plan. And then we're going to place that tag on a sheet. And I'm going to see if I can quickly do this in Revit here. So if I go back to the menu here and I pick these view tools, I'm going to say I'm going to create views from rooms. And it lists all of the rooms in the project. And I'm going to pick all of them and go create views. And it's going to create 93 views.

Now normally I wouldn't subject a live audience to watching a progress bar like this, but I'm sort of doing it on purpose to make a point. If you are the sorry individual that's been charged with making 93 views, this is sort of a aha moment for you. And again, particularly if you've

spent all day making these 93 views and somebody on your team told you that you did them wrong so tomorrow you get to make 93 more views. But you could see by sort of automating it here, I'm actually able to do this in something that's probably under a minute or something. So the moral of the story is, we're almost done here, it's worth investment in automation when you can save all of that time and frustration like that.

For some reason it changes views when I do that. Get back to Revit here. So I'm going to close this and then what it's done now, if I come down here and I look at my view list. It's made a whole bunch of these little views down here. And the one I like to look at is this one here. That's just because I happen to have put some furniture in there. So it went around and it cropped right around the room and made that view.

And then I won't put you through it all, but the next step would be to come in here and tag that view. And again we have an automated way of doing that. And what it would do is it would then add this tag to the view. And all this text down here is just a room tag that is placed on that room. So all of these values that we're seeing down here about the data about the room are actually just parameters of the room being expressed through labels in the room tag.

And then finally, actually what I want to do is show you a 3D version of a view. Just because you'll see in a minute we're going to use this when we go in to Access. So again, the automated process can also do this 3D section box clip and get a 3D view of the room. And then finally, with the tag view, we would go ahead and put that on a sheet. And so this is the tag view, just placed a title block on a sheet.

Now the process working in Access is sort of similar, or at least it starts out similar in that we make the views. But what we do instead of, well we take the views and we export them as Jpegs. And we're very careful about how we name the Jpegs so that when we get into the Access environment we can automatically link the views to be associated with a record in Access.

So this is a view of Access. And this, by the way, is a form. It's a data input form, and this is a form that you would use when you're working with your user groups, the client user groups, to gather the data. And then if I switch over to Access here, we can see what the actual room data sheet looks like. So here we have for each room we have the different views that we made and then you can see the data here on the left.

Now I'll get back to my PowerPoint here. There are a couple of other topics that I wanted to do

include but because we don't have enough time I just included them in the PowerPoint and they're also included in the handout. So I'll just mention what they are. The first one has to do with looking at the parent child relationships with families. For example, here if you're doing urban design and you're modeling property areas or property pieces with area objects. And you're building buildings with masses, which is nice because you can create mass floors and get the area of all of them.

One of the things you're going to want to do is calculate the floor area ratio. And this talks about how to do that. It's really difficult in Revit. And over on the right here you can see it's using a view filter to show a gradient so the different colors show the different floor area ratios. The second example here has to do with using related database tables and relating them to Revit colorfill plans.

And this is quite a complicated example because the data is broken over into a lot of separate tables. Plus, I'm trying to do some calculations to roll up the area so we a particular room type category on that floor. And to show it in the legend there. And you can see there's actually quite a lot of steps to this process that you can be grateful I'm not making you walk through. The end of the processes is that you have to get everything you want to see in the legend into a single text parameter and then you can organize the legend around that one parameter. So all the work goes into populating that one parameter and then Revit does the rest with its usual color plan technology.

The other thing that I'd like to show in closing here is an example from a real project. The sample data that I've been showing you is nice for development and it's nice for teaching, but it's a little bit misleading because real projects have a lot of complexity in them. And I got these courtesy of Katherine Chan at HDR. This is a real hospital. And if you look at this room data, and you look at the way she's numbering her rooms and so on, it's very, very complex. But it's very, very controlled. And that's what allows her to really keep her arms around that project when she's working with a lot of people and a lot of data.

So this is her version of the input form that she uses with the user groups. Not all of that data is going to go into Revit. That's part of the point of having a database. Some of it does, and here we see it being managed as parameters. I really like this example here. This is a report out of Access.

And as an architect, when you go into a meeting with a client you really need to control that

meeting in the sense that you don't want people to react to some stray element and have the meeting sort of go off in some direction that doesn't make any sense. And that's why architects tend to get really fussy about things like what the reports look like. How do they use colors, and fonts, and things like that.

And in this report you could see, for example, that she's done some subtotals, sort of in the middle of the report there. And down on the lower right there are some values that turn red when they go out of a certain allowable range. And these are all things that she's able to do because she really understands the data and she's working with the report writer in Access to customize the reports so that they speak exactly the way she wants to speak. And this is something that you really want to think about because you don't want to get into a situation where you're using an application that's sort of forcing you to produce reports that aren't working for you in terms of what you're trying to do.

This is her version of the room data sheet, and you can see the tremendous amount of data that she's managing there. And obviously you wouldn't want to have all of that in parameters in Access, or in Revit, rather. And then these final two reports are the summary reports that are to show the client that the project is, in fact, meeting the original requirements.

So with that, I'm going to conclude. And I want to thank you all for coming and for being patient with me. I would like to ask a favor, if you wouldn't mind filling out one of those speaker evaluation forms which I think are somewhere on the AU app, but I'm not quite sure how you do it. But if you can find it, there's the number of the class and my name. If you fill that out, it means that I can come to AU next year, so please do that.

The sample data that I was showing is in the class material. If you want to play with the White Feet tools you can download them and then use the sample data to play with it. I do work for Case, that's my day job. We're always happy to talk to you about this stuff so don't be bashful. My email is there, I'd love to hear from you if you just want to sort of toss ideas around or something like that. So thanks again.

[APPLAUSE]