OLIVIER DE WECK: So let's get going on system integration and interface management. So we're essentially moving up the right side of the V. And the idea is all the details have been designed. The concept has been selected. We know what we're doing, but now we have to re-integrate the system and make sure it works-- it performs the functions that are intended and satisfies the requirements.

And so what I'm going to cover today is three things. First of all, why do we care? Why is interface management important? Why is it important? There's two main reasons. One is because a lot of failures originate-- so a lot of failures originate at interfaces. And the other reason is because in order to design and manufacture a lot of systems, we need to work with partners and suppliers.

And I'll talk about interface management, sort of the details of it, the types of interfaces. I'll Introduce the DSM, the Design Structure Matrix, as an important method. And then we'll talk about ICDs, Interface Control Documents. They're a very big deal in practice. And then finally, I'll talk about system integration-- in particular, the sequence which you integrate systems, and then the role of standards.

All right, so here, the point that I want to make here is when you talk about interfaces, the first thing you need to be clear is, is it an internal interface or an external interface of the system? And so I think you remember this chart here that has-- is this you guys-- that has a system boundary that's defined. So the system boundary, in this case, we have this digital camera here.

And the internal interfaces are essentially the interfaces that are within this boundary. So those are internal interfaces within the camera, within the peripherals, that are included in the system boundary. And then we have external interfaces, which would be the interfaces between the system boundary and anything that's on the outside that you don't design or control directly. So that's an important distinction.

The next point I want to make is that you have to really carefully think about interfaces as seeds of potential system failures. And so I have three examples here for you. So the first one is-- this is actually-- Katya, sorry, this is from Russia, from 2007. This is traffic. This is one of the big-- not [RUSSIAN], but prospekt, the six lanes or eight lanes on one side, and there's a crossroad coming in here. This is a traffic intersection.

And you can see traffic is completely blocked on this side. So traffic engineers think a lot about interfaces, which are mainly your intersections, the roundabouts. Interfaces are really where a traffic system makes or breaks the traffic system. So that's a bottleneck. An interface is a bottleneck, basically.

The second example here in the middle, this is actually an example from MIT from an undergraduate class in design where the students were designing an airfoil. So this has been produced with a foam cutter. And you have a spar that goes through here. And this was intended for the MIT Formula SAE race car to create downward pressure on the rear axle.

They tested it in the wind tunnel, and it worked well at 20 miles per hour, 40 miles per hour. And then at 60 miles per hour, there was a failure, and the airfoil just basically disintegrated and was carried off into the wind tunnel and completely shredded by the propeller. Why did that happen? Well, they thought about the shear loads. So when you have an airfoil, there's a force that's acting orthogonally to the airfoil. So they thought about that.

The problem is there's also a torque, like a pitching moment, a torque that's generated. And the higher the air speed is, the higher the torque. They didn't think-- there was no real good way to react against the torque that was being generated. So at that high speed, the airfoil, all of a sudden, tilted and created a lot of drag and basically was destroyed. So the airfoil itself was OK, but the interface was not.

The third example here is a software example or a software interface, very famous. This was the launch failure of the Ariane 501. The very first launch of the Ariane 5 basically led to an auto-destruct of the rocket upon ascent. There was an accident investigation.

And I'm just quoting you here from the key passage, which said that "the active initial reference system transmitted essentially diagnostic information to the launcher's main computer." And so the main computer was interpreting this information as though the rocket has gone off course and trying to correct for it when, in fact, it hadn't. It was actually physically going exactly the right trajectory, but it was misinterpreted.

And this is also a very famous failure due to software reuse. So three examples of interfaces-- bottlenecks in traffic, structural failures, and erroneous function calls in software. And so be careful about interfaces.

The second reason we care is working with manufacturers and suppliers. So the picture here is essentially the Dreamliner, the Boeing 787. And you can see here the different colors essentially represent the different countries or the different suppliers that are providing different parts of the airplane.

Now, does anybody-- let me ask this question to MIT, see if we can hear them now. No? Well, maybe we can use the chat. What's different between the Dreamliner-- and you could have seen a similar picture, say, for the 737 or prior generation aircraft. So what's the real difference here with the interfaces, with the 787?

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Yeah, that's true. Composites is one thing, but that's a structural thing. I have something else in mind. There's some other fundamental change that was made in the 787 beyond composites.

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Nope.

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: This has to do with the way the airplane is assembled, with assembly.

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Yes, yes, exactly. So the supply chain is very different. And I guess I lost my-- hm, did I reverse this? So the point I want to make is that in the 787, the modules are essentially pre-outfitted. Like, all the systems, like the electrical system, the hydraulics, everything is already installed by the time the module arrives at the assembly plant in Everett and close to Seattle.

Before, you got the structural sections, but then all the electricals, all the inside was done on

site. So the modules are pre-outfitted with all the subsystems. And the final assembly itself is very short, only a few days. So a big, big step, and that's exactly right. That's the supply chain impact.

OK, so then the final point here is this is essentially a launch vehicle design, a launch vehicle that's being designed at NASA, the SLS. And if you think about a launch vehicle, you have the first stage, second stage, third stage, payload, the fairing. Actually, a lot of these fairings are made here in Switzerland.

And defining these interfaces very clearly is critical to reduce complexity. Understanding the different types, which we're going to get into next-- identifying the interfaces is a way to reduce risk. And I just talked about these problems originating at interfaces. And so hopefully, it's pretty clear now that this is a critical issue.

So what I'd like to do now-- and we'll try to troubleshoot the audio issues in the meantime-- is for you to turn to your partner exercise and share with each other, what is an instance in your past experience, maybe during an internship or in a project that you worked on, where interface definition and management was critical?

AUDIENCE: Some of us went this summer to Russia, to Moscow, to follow a summer camp on system engineering and space engineering. And we had a project there where we had to design a mission to deep space. And everyone was assigned to a group. The groups were around five to seven people, and there were five to seven groups.

And each group had a subsystem to design, but no group was assigned to system engineering. So actually, there was no one to make sure that all the interfaces were OK. So we had to go to each other group and ask for the interfaces.

And then when we had to go back, it was a mess. And at the end, every group had something good. But when we tried to put all the systems together, it was a big mess and nothing was working.

OLIVIER DE WECK: That's a pitch for systems engineering. So that's a good point. So even if you negotiate bilateral interfaces amongst subsystems, there's no guarantee that it will work at the system level. OK, can we get from MIT? Can somebody give us what you discussed?

AUDIENCE: Is this the mic working?

OLIVIER DE WECK: Yeah, now we can hear you.

AUDIENCE: Great. So my grad school project here at MIT is a secondary payload on a big NASA mission. And as a secondary payload, the interface control is a really big deal. So we're getting all of our power and all of our communication from the spacecraft, and we're also getting a heater circuit from them. So any change is a big deal. We had to change our heater circuit and we had to go through a lot of paperwork just to splice a wire. So no issue with it, but it's not as easy as it sounds, and there's a lot of control over it, I would say.

OLIVIER DE WECK: OK, maybe another example from MIT. Somebody else?

AUDIENCE: Does this work?

OLIVIER DE WECK: Yeah.

AUDIENCE: So the--

OLIVIER DE WECK: We've got it fixed out now, by the way. I think this is all working now, so good.

AUDIENCE: The last project I was a part of was building an aircraft for a Red Bull Flugtag. So it was like a giant glider that we'd push off a platform.

But we were kind of building it in a hurry. The time constraints were pretty quick, and we didn't really have all the requirements until a month or two before the competition. So we winged a lot of the design and construction. And so we thought about how to build the wing and how to build the empennage and how to build the cockpit, but we didn't think about how to put them all together.

And so it turned out just being wrapped with a lot of carbon fiber and epoxied. And in some cases, we used wood. And it was a very heavy and poor-looking assembly once all that was together. Even though each part individually looked great, they didn't really come together perfectly. So we'll change that next year, but could have thought more.

OLIVIER DE OK, good. Great example. All right, so hopefully, this is sort of motivation that, in a sense, a

WECK: system is the functions you want, the components, and the interfaces. You've got to have all three. If you only have great components but you don't understand the interfaces, you stick them together, I'm not sure what you're going to get. Yes, do you want to--

AUDIENCE: [INAUDIBLE] Is it also important the interfaces with testing facilities?

OLIVIER DE WECK: Yeah, absolutely. So testability requirements, how you're going to test it, is critical. Also, ground support equipment-- you know, like maintenance equipment. So not just the interfaces, the system when you're operating it, but during testing, during maintenance. And those are often forgotten as well, so great point.

So let's keep moving here. I want to talk about the types of interfaces, and then we'll get to DSM. So here's a set of examples of interfaces that we encounter. Let's see if we can get the slides back up here.

So essentially, here's some examples. In the upper left, we have a valve, and then a tank on the right side. And we essentially have mass flow from the valve to the tank, and m dot, dm dt. We have a rocket. If you think about the rocket interface with the payload, it's transferring momentum, right? The rocket provides thrust, force F, for some amount of time, delta-t, and that's the momentum imparted.

We have a heat exchanger that is maybe giving heat off to the air, the surrounding air. So we have a heat flux, q dot. We have solar cells, PV cells, that are producing electrical power, voltage times current. And this power flux is being sent to a battery where the energy is stored. So these are all pretty obvious examples.

And then on the right side, things get maybe a little bit less obvious. You go to a URL and you download an HTML file into your browser. That's an information. You're getting data. You have a motion sensor attached, say, to your house, and it triggers an alarm. That's a command, different from data.

NPR is National Public Radio. This is kind of the high-quality radio news reporting. And you listen to this. So there's a cognitive interface. And then if any of you have had any psychotherapy, this is also an interface between the patient and the psychotherapist. This is an effective interface.

So these are very different. And you can say, well, there's hundreds of types of interfaces. Well, no. The argument here is all interfaces can be reduced or described by these four

canonical types.

The first one is physical connection. So A connects to B. B connects to A. A physical connection typically implies that there's a force or torque that can be transmitted across the interface, and it's always symmetrical.

And then we have energy, mass, and information flows. And I'll give you some examples of each of these types of interfaces. And these energy, mass and information flows typically are asymmetric. It flows from one to the other. Physical connection is always symmetric.

So let's go through these, and I'll give you examples. And then we'll see what can we do with it. So here's physical connection examples-- rollers, brake pads. A finger touching a touch screen is a physical interface.

The interface can be reversible. In other words, the connection is temporary-- and examples of that would be electrical connectors, USB ports, latch mechanisms, nuts and bolts-- or permanently connected, such as in-- so one of you mentioned riveting, spot welding, fusing. And a fun question that comes up, well, I'm a software engineer. What is physical connection mean for software?

Well, it's a kind of tricky question. I think the closest analog we have is compiling. If you have pieces of source code and you compile your program, then you've essentially fused this together into something new, right? So I'll show you the example.

So how do we describe it? Well, here's an OPM model of-- I'll give you a couple examples from xerography. So we have a main motor here, number one, and then two different clutches, number two and number three. And the process linking them is engaging, right? The main motor engages through the clutch.

We can hide the process, and then we just have this bi-directional interface between one and two and one and three. And then if we show this as a mini-DSM here, one connected to two, one connected to three, but two and three do not connect directly with each other. They don't have a direct physical interface. And you can see that because those cells are empty. So physical connection implies symmetry in the design structure matrix that we can build up. Yes, [INAUDIBLE]?

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Oh, DSM-- so DSM means Design Structure Matrix. And I will introduce that in a minute. We've briefly mentioned it before in the class, but I'll get into it in some detail.

Here's some more picture examples of physical connections-- so a welding joint, irreversible, reversible, nuts and bolts, RJ45, an electrical connector. And then here's your-- hey, I'm an old guy. I learned Fortran. Well, Basic was my first language, and then Fortran. I did a lot of Fortran. You guys probably don't even know what that is. But Fortran-- right, [INAUDIBLE]? Fortran's kind of a big deal still in some--

AUDIENCE: [INAUDIBLE].

OLIVIER DE WECK: OK. So basically the way this works is you have your source code, which are .f files. These are basically Ascii files, right? And then first, you transform them into binary files, object files. And then these get merged together into an executable. So that's the process of compiling.

Any questions about physical connections, this type of interface, physical connection? Is it clear? OK, any questions at MIT? Can you still hear me?

AUDIENCE: Yes, we're good.

OLIVIER DE WECK: OK, good. So let's move on to energy flows. Energy flows are present when there's a net exchange of work between two components. So power is flowing across the interface dw dt, right-- joules per second, watts. And the energy, this energy flow, this power flow, can take different forms-- electrical power, which is very common in many products. And so the electrical power usually comes either as DC, Direct Current, or AC, Alternating Current.

And if you know a little bit about the history of electrical systems and networks, there was a huge war. It's called the AC/DC wars. It's not the band. This is not the rock music. This is the war between Edison, who is a big proponent of direct current, and Tesla and Westinghouse, who were really pushing AC for big power distribution.

So we won in the end? AC won. It looked initially like Edison was really-- and there's still, in New York City, there, for a long time, and other places, you still had DC power systems. The problem with DC power is there's huge losses with distance. And with AC, you can go to very high voltages and transmit power with relatively little loss over large distance. That's the main reason that AC won out, essentially.

The thing that's important here-- and power, of course, is current times voltage. The thing

that's important is that these voltage levels-- and in the case of AC power also, the frequency is very important, 50 or 60 hertz. This is very standardized, OK?

So if you're going to design a satellite system, or a rover, or an automotive system, or a medical device or a consumer product, the voltage levels are pretty-- there's a few choices, but you can't just choose some arbitrary number, because your whole industry, your batteries, your connectors, everything is now pretty standardized.

So for the Octanus 1, Rafael, what did you guys choose for your power system? What's your--

AUDIENCE: [INAUDIBLE] 3.3 volts.

OLIVIER DE WECK: 3.3-volt bus across, OK. And then if you need more energy, you just have to add batteries. But that will essentially-- and you can do transformations as well. You can have different voltage levels in your system, but that adds to the complexity.

Thermal flux is essentially heat flux, dq dt. There are three fundamental mechanisms of heat transfer-- conduction, convection, and radiation. In spacecraft, radiation is probably the most important because that's the only way to get rid of heat out of your spacecraft. And then you can do conduction and convection internally.

RF power, so microwaves. Transmitting or transferring power through microwaves is becoming-- of course, we now had microwave ovens for a long time. But even doing power beaming over larger distances with microwaves is definitely something that's been demonstrated, and there's a lot of interest in it. Here also, we are pretty standard frequencies, so 2.4 gigahertz, 5.8 gigahertz. You have a whole industry built around these standards.

By the way, why the 2.4 gigahertz? Let's see if-- at MIT. This 2.4 gigahertz for microwaves, where does that come from? Anybody know where that comes from?

AUDIENCE: Is that the frequency that can be transmitted easily through the atmosphere?

OLIVIER DE WECK: Actually, no, it's not good. It's absorbed. It's absorbed in the atmosphere. It's the opposite. This 2.4 is very important because it's a resonant frequency of the water molecule.

So your microwave oven will work at 2.4 gigahertz. Why? Because if you radiate microwaves at that frequency, it makes the water molecules vibrate because that's a resonant frequency, and you transfer a lot of heat or a lot of energy into your water molecules, which are the

majority of your food.

So it's actually really bad for-- you'd like to use 2.4 gigahertz, but if you try to use that frequency for atmospheric power transfer, you're right at the resonant mode. So if you have a high level of humidity in the air, you're going to lose a lot of that power, just heating the atmosphere up and not transmitting the power. Does that make sense?

AUDIENCE: Yeah, thanks.

OLIVIER DE WECK: OK. And then another very popular frequency is 5.8 gigahertz. And then we can go to X band, which is closer to 8 gigahertz, et cetera. But anyway, so this is a kind of growing area is microwave power transfer.

And then of course, the more old-fashioned, perhaps, but still very important is mechanical power, so transferring energy and power through a mechanical interface. So if it's linear, it's force times velocity. If it's rotary, it's torque times the angular rate of rotation.

And then finally, energy-- typically, if you want to have energy flow, it does imply that there's a physical connection as well, like some kind of wires or a gearbox or something like this. However, a lot of interest now in wireless power transfer. But it's still kind of a-- for it to really work reliably at high power levels, this is pretty immature technology still.

So here's some examples of-- again, this is from xerography. And you'll understand why I'm using these examples in a minute. So we have a paper. We have toner, unfused toner. And we want to fuse those two together. We do this through a heat roller and heat and a belt.

And essentially, we're transferring energy from the heat roll to the paper, which allows it to fuse the toner. So we have transfer of energy from one to two, which is shown by this green mark here in the DSM. So heat energy is transferred from system one to system two.

One question that often comes up is, well, what if we have like waste heat? There's heat losses, and there's parasitic energy flows. Do we care about those? Do we represent those?

And the answer is yes, you should, especially if they affect your performance or your requirements. So the first thing you typically do, as you map out your interfaces, is you map the flows that you want to happen, that need to happen. And then on top of that, you map out the flows that happen even though you didn't design for that, but you need to account for them. So heat losses would be an example of that waste heat flux.

Any questions about energy or power flows? MIT, any questions, energy or power flux?

AUDIENCE: We're good, thanks.

OLIVIER DE WECK: OK. Next one, mass flows. Mass flows is when matter, physical matter, is being exchanged between two elements or subsystems. So mass flows is dm dt, kilograms per second. And the form of the matter can be fluids, gases, solids. I guess plasma as well.

There's a big plasma center here on campus. But the primary three are fluids-- so if it's fluids, it could be like a cooling liquid, a refrigerant, fuel, water. If it's gases, air, exhaust gas. And then for solids, examples would be toner, paper. If you're in the mining industry, it's iron ore.

And typically, mass flow implies an underlying physical connection, like some kind of channel or conduit. So Rafael, your example of your sucking up and melting the snow and you said there's a hose, so the hose is itself a component, but it enables the mass flow of the melted snow into some holding tank or reservoir. And so the hose is the physical connection. And then the mass flow that happens through that hose is what's shown here.

Mass flows can be open. So there's a source and there's a sink. The source and the sink could be inside or outside your system boundary. You have to think about it. But also, mass flows can form continuous loops, particularly if you recycle things. Like for example, in the refrigerator, your cooling liquid continuously cycles through your system.

So here's some examples of mass flows. We have a heat exchanger. This is a U-type heat exchanger. So on the top, we have what's called here the shell side. A fluid is coming in and is running out on this end. And then we have horizontally, a fluid that's coming in on the lower side, cycling through, exchanging the heat, and moving out.

So a heat exchanger is interesting because what kind of flows happen in a heat exchanger of the flows we've discussed so far? Energy and mass flow, right? So the two of them happen together. In order for the energy flow to happen or transfer to happen, the mass flow has to happen as well. So these two flows are tied to each other.

Here's an example from xerography. So we have our photoreceptor belt. We have residual toner on the photoreceptor that needs to be removed from the photoreceptor. And we do this through a cleaning lamp, and then a blade. So in this case, you can show this as an OPM, and then you can show it here where that residual toner is transferred from photoreceptor one to

the cleaning blade two. And this is shown in the DSM as a mass flow from one to two.

And then we have one more, which is the information flows. So this is really where the big revolution has happened in the last two decades since a lot of functions that used to be done mechanically have been replaced with software. And here's some examples of these information flows.

So if your system has a user interface, a GUI, a graphical user interface, or some kind of I/O, input/output, with the user, that's critical. That's an information interface. And so Rafael, again coming to your presentation-- so I'm going to pick on you a lot today. Not pick on you, but I'm going to cite you a lot.

You said this rover is going to be controlled somehow, right? And the big question is always, how much autonomy do you put inside the thing, and how much do you rely on a user that's remote? And that's clearly an information interface.

The nature of the information, is it analog? Is it digital? Is it wireless? So on the analog side, we're essentially talking about some signal, some voltage, and there's been a lot of work done on ADC, analog-to-digital conversion. And then the opposite is digital to analog. And depending on how many bits you have available to you, you're going to have some discretization errors.

A purely digital interface, we refer to it as DIO, Digital Input/Output. And then of course, wireless, this is the standard, the 802.11 standard, which is really very important these days. If you think about why we need all this, mainly, the main reason is control, right? We have sensors, actuators, controllers. And because this information is never perfect-- there's gaps in the information or there's noise superimposed-- we have to also filter and amplify the information.

And so if you think about the type of information flows-- for example, in spacecraft, we could have telemetry, sensor data. And telemetry fundamentally means, I want to know how is my system doing. Is it healthy? Are all the subsystems working? Are there some anomalies? Are components within their temperature limits? That's what we call telemetry.

And then commands are when you are sending a command to the system and say, you know, turn this way, turn that way. And fundamentally, telemetry and commands are treated quite differently. So where do you need better quality information? Where is the link more important,

on the telemetry side or sending commands? What do you think?

Actually, no. So if your telemetry is imperfect-- I mean, it depends a little bit what you do with it. But the commands are typically more critical. Like, if you have a bit error in a command, and it says fire the engine but you didn't want to fire the engine, you can lose your mission. So typically, what's known as the bit error rate has to be lower, meaning the link budget is much better for commands. So there's an asymmetry.

OK, so here's some examples. So here's a typical control loop. You have your plants, your system, your actuator, sensors, a comparator that compares your reference signal-- this is how you want the system to behave-- and sends that to a controller, which then closes the loop. The example here, again from xerography, is you have some original document. You want to make a copy here.

So there's a lamp that shines light onto your document. It goes through a lens, a laser diode, and then to your marking system. The OPM, you have your original, your optical system. You create a digital image file which then is sent to the marking system. So if we simplify all this, we can say there's information flow from the optical system to the marking system. From one to two, information is transferred.

So any questions? So those are the four fundamental types of interfaces and examples with that. And so the theory, what the theory of design tells us is that this is universal. Any kind of interface falls into one of these types. And in fact, even in biological systems that's true. Yeah?

AUDIENCE: When we have a system [INAUDIBLE]?

OLIVIER DE WECK: I'll show that to you in the DSM. You actually track them separately, but they're within the same-- it's one big interface, but it has these different subflows within it, which means that-- it's not just the fact that there is an interface, but the complexity of the interface is important.

So the most complex interfaces obviously would be the ones where you have all four types, right? It's a physical connection, and there's an information flow, and there's an energy flow, and there's a mass flow across the interface. Usually, it's only a subset of those, but you can have interfaces where you have all these four present at once.

Any questions about these types of interfaces at MIT?

AUDIENCE: No, we're good. Thank you.

OLIVIER DE WECK: OK. In general, can you hear better now? Is it pretty stable over there?

AUDIENCE: Yeah.

OLIVIER DE WECK: Good, all right. So let me talk about DSM. [INAUDIBLE] was asking about DSM. So DSM is a fairly recent method, the last couple of decades, for really making a map of your system that shows explicitly these interfaces.

So here's a very simple example. This is a sample system with five components. It has a pump, a controller, a motor, a valve, a filter. And in this case, we have mass flow from the pump through the valve into the filter.

And so this block diagram on the upper left, the exact same information is contained in this matrix. So a DSM is always a square matrix where your components are the rows and columns, equally labeled. And the diagonals here, there's no information on the diagonals. So the component connects with itself, obviously. So you keep the diagonals sort of blank.

And then all of those interfaces and flows we mentioned are shown as off-diagonals. You have to be a little careful because there's two definitions of DSM depending on the inflows and then the outflows. So the definition that we mainly use in North America is where the inflows into a component are horizontal. So the controller sends information to the valve. This is that blue square right there. And you can see that's shown here on the block diagram.

But the valve also sends information back to the controller. And that information would probably be, valve should open or close. And when the valve is actually opened, it would confirm that back to the controller that the status of the valve is open or closed.

And then we have the energy flows. The numbers that are shown here, I don't want to get too much into this, but there's a really cool binary scheme for putting a number that you can then have one number in that cell.

And based on that, you can, through the binary code, reverse engineer what kind of interface is it. Is it just physical? Is it just binary? It's a kind of trick for replacing these colors and cells with just a number that allows you to then reverse engineer what kind of interface is it. So if it's empty, it means there's no direct connection. And then you have mechanical, mass flows, information, and energy flows.

So let me show you this on a real example in a minute. So this method, design structure matrix method, there's some synonyms for it-- design dependency matrix, n-squared matrix, n-squared diagram. In matrix or graph theory, it's called an adjacency matrix. Don't be fooled; it's the same thing. It's just a square matrix that shows you what connects to what.

And then the simplest one is just a binary, 0's and 1's. But the richer DSMs, they contain more information about the type of interface. So a lot of work was done originally by Don Steward on this. This is an original reference from '81. And then at MIT, Professor Steve Eppinger at the Sloan School has done a lot of work on DSMs.

So fundamentally, it's a matrix representation of your product architecture. Most of the literature you find just uses the simplest, this binary DSM. But I'm actually a fan of adding information into the DSM that tells you about the physics of the interface as well.

So let's look at an example here. Do you remember the refrigerator case study we did a few sessions ago? So this is a liaison diagram, level one decomposition. So the refrigerator's been decomposed into 10 components-- the door, the condenser, power supply, all the way to the compressor. And all we're doing here is saying what is physically connected to what.

And the reason it's called a liaison diagram is because it's going to help you think about the assembly sequence of how this should be put together. So another way to think about this is like the skeleton. This is the skeleton of your system.

So if you do have a liaison diagram, shown again-- this is the same one on the left here-- you can just translate that to a matrix. The matrix, this DSM here, I'm using X's instead of 1's. But these two contain exactly the same information. One is a graph view, and the other is a matrix view, but they're identical-- the same thing, the same information, just presented in a different way. And of course, if you pay close attention to the matrix, you'll see it's actually symmetric.

So then we can say, well, what kinds of interfaces? And here come the four types that we just discussed-- the physical connection, energy flow, mass flow, and information flow. So if we apply that idea to the refrigerator, here's our 10 by 10 physical connections. So we have a design scripture matrix with only the black marks. So it's symmetric. And this reflects the liaison diagram.

And then we can add, on top of that, the mass flows, which are in red. So the mass flows, the way you read this is we have a mass flow from the compressor to the condenser, and then

from the condenser to the evaporator, and then from the evaporator back to the compressor. Do you see that?

And we could rearrange these to be closer to each other. We could regroup the DSM. But essentially, these red marks represent the refrigerant cycle within the refrigerator, within the machine. And so you can check by tracing these from row to column. You can actually check whether the loop is properly closed.

So then energy flows, since the refrigerant loop that I just talked to you about is important, you can see that those connections we just talked about always also carry a energy flow with them. And then finally, information flow is very little. So in this case, there's only one that I've represented, which is the flow from the thermostat to the power supply.

The thermostat defines the temperature at which you want to hold the refrigerator. And if temperature goes up, the power supply kicks in and drives the compressor. And as soon as the temperature falls below some threshold, then it reverses, and the compressor turns off. So when your refrigerator kicks in, that's an information flow from the thermostat to the compressor to the power supply.

If you own one of those fancy, new Samsung refrigerators where the door has a flat screen TV in it and you can push buttons, there's a lot more blue here. There would be a lot more blue in a really high-end refrigerator these days. OK, any-- yes, please.

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Yeah, so the question-- I'm just going to repeat the question-- is, can you also map the external interfaces? And the answer is, absolutely. So what you have to do, though, then is to kind of draw a bigger box around this and add here the external elements-- so the human user, the wall outlet, the grocery store, wherever you're including in your external world that's interfacing with it.

You would have to represent at least those elements that have a direct connection with any of it across the system boundary. You would add them here, and then show-- but you would have to then put a big box around it to make it clear what's inside the system and what's outside.

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: That's correct. That's correct. And we can argue-- of course for the refrigerator, the main interface that you typically have-- well, there are two. Let's say we go for what you're saying. We're going to add some external interfaces. For a refrigerator, what are the two or three most important?

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Correct. So where would that enter here?

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: The power supply, right? So we would add, like, wall AC power. If we added that, then in this cell corresponding to that, you would have an inflow of power. And then--

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: So this is the power supply inside of the refrigerator, inside the system boundary. But if you had the wall outlet, then that would be an external. And then you would have an interface across, but into the power supply.

So that's one. What's the other one? Do you remember we talked about the Carnot cycle? Well, yes, what do you have to do with the heat?

AUDIENCE: Burn?

OLIVIER DE WECK: Yeah, so the heat has to be removed, right, radiated out. And where does that happen? Which element here is where the heat gets removed, which one of these 10?

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Not really. The refrigerant does the internal heat transfer, but the heat transfer to the outside, to the air. Refrigerators don't work well in vacuum chambers. Remember that.

It would be the evaporator. It's at the evaporator that you're radiating the heat to the surrounding air, the environment. So you'd have the surrounding air as an external system, and then and energy transfer from the evaporator to the outside. But it's a good question. So you have to always define your system boundary.

So the question then is, when do you generate these DSMs? And fundamentally, there's two

ways, top-down or bottom-up. So top-down, you start with a model of your system. And we talked about those system modeling languages, like SysML, or in this case, OPM. And then you hide some things. You hide attributes and states. You collapse all your processes into these tagged structural links, and then you generate your DSM.

And then the other way to do it is bottom-up where you already have a design or a system. You decompose it. For example, you do a product dissection. You do your bill of materials or your parts list. Then you do your liaison diagram. You show the physical connections.

And then on top of those physical connections, you map your mass, energy and information flows. And once you have that, then you can start manipulating the DSM. You can group components differently. You can show modules and so forth.

So I want to show you a real world-- this is not just sort of an academic example, but this is a real-world example of a DSM that we built for a research project with Xerox. The machine that's represented here is in the upper right. This is the iGen3 digital printing press.

And represented here is its architecture, its interfaces, as an 84 by 84 DSM. So the machine has been decomposed into 84, really, subsystems, because you have a lot more than 84 parts. But at this level of abstraction of 84 by 84, we have a total of 572 physical connections, 45 mass flows, 167 energy flows, and 165 information flows.

And there's a little more detail here in terms of the types of flows. So in red, we have mass flows-- paper, toner, air, ozone and dirt, because you've got to keep dirt out of the-- the printing engine is shown here in the middle. Let me just show this. So you're probably scratching your head now. Ozone, really? Ozone? Why do we keep track of ozone? What do you think? Why do we care about ozone?

AUDIENCE: [INAUDIBLE] at ground level.

OLIVIER DE WECK: It's a pollutant at ground level. And where does the pollutant come from?

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: So it's generated inside the machine. The reason is these machines use very high voltages. Xerographic processes use pretty high voltages. So if you apply a very high voltage, you actually generate ozone through a-- it's an electrochemical reaction that happens inside the

machine.

Nobody used to care about this until people working in these print shops started to feel symptoms and started to complain. So in the US, it's called OSHA. That's the Occupation Safety and Health Administration. They came in, and they took some measurements. And wow, the ozone level is through the roof. It's multiple times above the allowable maximum limit. And the reason is because of this kind of xerographic process.

So now, these machines have to control the ozone. And essentially, this box here, the printing engine, is sealed. It has its own air conditioning system to prevent the ozone from leaking out. It's a great example of how occupational safety drove new requirements which then increased the complexity of the machine. So we keep track of the ozone flows.

And then in green, we have the usual suspects-- high voltage, low voltage, DC, mechanical energy, and then heat energy. So there's some coding that goes on here. So in total, we have 1,033 non-empty cells here, which is about 3.7%, so about 4%. This DSM has a density of about 4%. And that doesn't sound very high. But believe me, this is a very complex machine.

This DSM that you see here is version 31. So it took 31 versions to create this DSM. And so why did it take that long? Because this was published as an official document capturing the architecture of this machine.

So what that means in practice is of these 84 components or subsystems, every one of those components or subsystems is owned by a different person or a different team. And we went through a process where all of these 84 teams, for their component, had to sign off and say, yep, you've accurately captured my interfaces. My physical, mass, information and energy interfaces have been correctly captured in this matrix.

And oh, by the way, the empty cells have also been certified. So if there's an empty cell here, there's no question whether, well, maybe it's just somebody forgot to put the interface there. No. The owner of that row and column certified that there is no direct interface between this component and the other component. That's why it took 31 versions.

But now that you have this map, you can do a lot of things with it. You can do architectural analysis. You can benchmark against other products. You can do what we call technology infusion analysis, which was done.

You can say, well, what if we add this feature in the machine? OK, well, where is that feature going to go? What are the interfaces of that new feature? Do we have to change the software? Do we have to physically integrate it?

This is a fantastic, very powerful tool for showing the interfaces and managing them. Any questions about this DSM approach? The real way to do this is to actually build one yourself. And you can see it's a lot of work, but once you have it, it's very powerful. Yes, [INAUDIBLE]?

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: So there's templates, like Excel templates. But there's also software. There's a website called DSMweb.org. And I'm happy to send that out for you as a resource.

There's even yearly conferences. There are DSM conferences where people get together from academia, from different companies-- like BMW, for example. I'm a big fan of BMW. I think they do a great job. Their system engineering is very, very strong. They use this on all their vehicles, on all their subsystems.

And so this DSMweb.org, you'll see sort of the software that's available and so forth. There's also some commercial software available. So yeah, please, go ahead.

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Ah. So let's think about this. The 7 plus or minus 2, how does the 7 plus or minus 2 relate to the 84? This is an 84 by 84 DSM. So how do the two relate to each other?

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Right. But what's the relationship of that rule with the 84 by 84?

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Yeah, so this machine might have-- if you really want every little detail, this is probably a level 4, level 5, right? But what's shown here, this is essentially the equivalent of a level two representation. So do you remember 7 plus or minus 2 squared is somewhere between 25 at the low end and 81. So we're just slightly above this.

So my argument here is this is essentially a level two DSM where you decompose the system

down two levels, and then you show the DSM the interfaces at this level of abstraction. You could go level three, but now you're going to have a 500 by 500 DSM, which is almost impossible for a human to really understand and get any use out of it.

There's also been some work in hierarchical DSMs. So for example, if you look at the feeder system-- you see in the upper left here that says "feeder"? Well, what is the feeder? It's all this upfront stuff that feeds your paper and your media into the print engine.

So you could take-- this is just two rows here. You could take those and break those into much greater detail. So you'd have-- kind of like you guys did the requirements tree, the requirements hierarchy. You can have a hierarchy of DSMs. I'm not showing that here, but that's the relationship. This is essentially a level two decomposition DSM. Good question.

MIT, any questions?

AUDIENCE: Yeah, I just have one question about the liaison diagram.

OLIVIER DE WECK: Yeah?

AUDIENCE: Is there, I guess, any meaning or information that comes from the distances between-- like when you draw it, the way that you draw it?

OLIVIER DE WECK: No, there really isn't. So the tricky thing is if it's a very complex system, like if we did the liaison diagram for the 84 by 84-- let me just go back here-- it would be pretty messy. And so you want to sort of do the layout of the liaison diagram so you minimize the number of crossing lines, for example.

There's actually some graph visualization algorithms that try to create as clean as possible of a graph. One of those is-- it's called a spring energy algorithm. It basically treats these links as though they were springs. And then depending on how far things are, there's spring energy.

And you want to minimize the total energy of the system. And by doing that, usually, the graph looks pretty clean. But there's no inherent meaning in how you lay it out. Does that makes sense, Sam?

AUDIENCE: Yeah, thank you.

AUDIENCE: I have a question too--

OLIVIER DE WECK: Go ahead.

AUDIENCE: --about the liaison diagrams. I was doing something similar, at least where I had each block as a functional aspect of the system. It was for an avionics system. And I found that I needed to, like, make power a separate thing because it was more of like a tree. We had one power distribution system.

But the thing I wanted to bring up was the information flow. Originally I just had one color for that, but I didn't find that too useful. What I plan on doing is splitting it up into command and telemetry. So is drawing information just as one entity typically done? Because I think it's more useful to have command and telemetry, or like data and commands.

OLIVIER DE WECK: So you know, that's fine. And depending on what you're going to use it for, you can redefine the colors. There's no global standard for how to properly do a DSM. That's still kind of evolving.

So if you have different types-- and you saw in the Xerox example, we used letters as well for the different types of mass flows-- P for paper, T for toner. So the way you're going to code it is really up to you. There's no global, right one way to do it yet. So that sounds good. Actually, you did the right thing there.

AUDIENCE: Cool, thanks.

OLIVIER DE WECK: How are we doing time-wise?

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: OK. Any other questions? Yes?

AUDIENCE: What kind of [INAUDIBLE]?

OLIVIER DE WECK: So why are we doing this?

AUDIENCE: Yes, exactly.

OLIVIER DE WECK: Well, pretty pictures. No, I'm sort of half-joking. Part of it is definitely a graphical, visual, powerful way to convey the whole system architecture. If you think about it, there aren't too many ways of actually visualizing the whole system. This is a very abstract view.

But there's a-- and then this DSM here at Xerox, it printed out as a huge, wall-sized poster. And you put it on the wall. And people actually go up there and say, ah, that's my subsystem. And then they argue and so forth. So part of it is definitely a visualization.

But the other thing you can do with it, particularly if you start putting the numbers in, you can use this for complexity quantification. So you can quantify the complexity of your system and then compare it against a competing system or a prior generation system.

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Correct.

AUDIENCE: You could not have a macro view on it.

OLIVIER DE WECK: Right. So this is a macro view, but if you really-- then you need to put numbers for the component complexities, the interface complexities and so forth. And there's quite a bit of research in this. So you can then move this from a purely visual tool to becoming an analytic tool. Yes?

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Yeah. And one thing we'll talk about next is interface control documents as an important-- so in theory, in theory, every one of these interfaces, you can have a document or a digital file if you click on this. And this is where the tools are not fully mature yet. But you'd like to be able to click on this, and then it links you to another document.

And what is that other document? It's the ICD, the Interface Control Document that defines that interface. And then you have a lot of details. Like if it's an electrical connector, how many pins, what's the function of each pin? And that's really where things become useful, OK?

All right, let me move on. But I did upload for you guys. So it's really up to you if you want to do a DSM at your PDR. It would be nice, but I don't absolutely-- if you like this, if you think it's useful, try it out for the PDR. But it's not a must-have requirement. But in the full-year SDM

class that we teach, we're just doing that right now.

And so you can use it for reverse engineering the architecture of a software package. You like open source software. You just download it. You generate a DSM, and you can start seeing what are the different modules in the code, what do they do, how do they talk to each other. It's incredibly powerful.

But what I did do is I uploaded for you guys-- I said there's a top-down, and then there's a reverse engineering or bottom-up way of doing it. This is a 17-step procedure, pretty detailed, a document that I uploaded under the reading section. So if you go on the website, under the reading section, you will find-- this is like a four or five-page document that says step by step how do you do a DSM.

And again, so at a high level, then, what you get out of it is-- here, I've sort of hidden the details from what you saw before-- is the high-level architecture you can see. So in the case of the Xerox example, we have our frontend system, the feeder system, the main marking engine. You can see the modules. And then you have the finishing system down here.

And so many systems and products have similar architectures. OK, any questions before I move on. Let's talk about ICDs, Interface Control Documents. And in the system engineering engine from the handbook, this is right here in the middle.

This is the technical management processes. Number 12, interface management. It's a huge deal. If you don't do it well, if you don't do interface management well-- and we heard some examples-- it's really trouble.

So the purpose is for controlling your development effort. It's really important-- I just highlighted a few words here-- if you have geographically diverse technical teams. And maintaining interface definition, once you've decomposed your system, is absolutely critical.

So let me talk briefly about the different-- so this is a little bit more like traditional system engineering where you define the interface with a document. We've said that system engineering is moving to become more model-based. We don't want big piles of paper anymore. But the reality is that many projects and programs still have a lot of these documents.

Ideally, you'd just have a model, a digital model of the interface. Then you don't need a separate document. But if you are working with suppliers or contractors that don't have the

same modeling capability as you have, these documents are critical.

So the first one, it's called IRD, Interface Requirements Document. And this is-- do you remember when you did the requirements analysis in assignment two? This is one of the six classes of requirements is interface requirements. So that's the idea, is all the interface requirements are pulled together into an IRD.

It defines the functional, performance, electrical, environmental, human, physical requirements and constraints that exist at the common boundary between two or more functions or system elements. So you haven't designed the interface yet, physically, but you've defined what the interface should be able to do, its functionality.

Then we have-- this is the most common. You've probably heard this acronym before-- is ICDs, Interface Control Document or Interface Control Drawing. And so that is essentially the specification of the interface.

What does the interface look like? So if you look at a ICD of an electrical connector, you'll see the pin-out, you see the voltage levels, you'll see the purpose of each signal for each pin. Maybe there's some pins that are unused that are reserved for future use. That's also very clearly defined.

And the key about the ICD is it's a two-sided document. So there's a subsystem A and a subsystem B, and they share a common interface. And the ICD, the Interface Control Document, will define both sides of the interface as opposed to IDD, Interface Definition Document, which is only a one-sided or unilateral document.

So for example, let's say your company makes a very fancy camera or sensor to be used on a spacecraft, but you don't know who is going to use that camera, how are they going to use it. So what you're defining is essentially your side of the interface-- physical, functional. And then it's up to whoever your customer is to take the IDD and turn it into an ICD. But the IDD is only one-sided, unilateral. And the ICD is bilateral, interface definition. Does that make sense? OK.

So here's-- again, this is from the handbook, the system engineering handbook-- interface management. On the input side, you start with your interface requirements, any changes to the interface that come up during the lifecycle of the project. You go through the steps and out come your interface control documents, any requirements changes, and then interface management work products, which might be other supporting documents.

For example, a user manual or an operator manual that will actually tell you how to properly use the interface would be an example of a supporting document. OK, any questions about that before we get to system integration?

So who's actually read an ICD or had an actual ICD in their hands? You have, yeah? [INAUDIBLE], I'm sure you've had that many times. Can you tell us what it was?

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Let me give you the mic. So-- [INAUDIBLE]

AUDIENCE: So we were working on a project where a Chinese asked us to contribute a camera payload for their satellites. And so we had to define all the electrical and data interfaces. I was in charge of the data interfaces.

And there were design documents saying how will the interaction work between the two systems. Let's see. Because [INAUDIBLE] protocol, I had to tell the satellites what to do if the camera fails and how should the camera fail if there is a problem on the computer, so all the failure cases and this kind of thing.

OLIVIER DE WECK: So it sounds like [INAUDIBLE].

AUDIENCE: Yeah, it's an ICD.

OLIVIER DE WECK: Good. And was it used then, or--

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Great, excellent. Any examples of an ICD or IDD or IRD on the MIT side? Anybody have experience with this?

AUDIENCE: No.

OLIVIER DE WECK: Was that you, your honor?

AUDIENCE: Yes, I personally have experience with ICDs, but no one in the room does.

OLIVIER DE WECK: OK, well, why don't you tell us about yours? Because I know you worked on this gravity gradient sensor.

AUDIENCE: Yes, exactly.

OLIVIER DE WECK: So maybe that example. Yeah, why don't you tell us about that?

AUDIENCE: So each system had its own ICD. And that was primarily because the rest of the team needed to understand the inputs and outputs and how the specific system works. So it not only works for system integration. It just works for the team as a whole in its understanding of the project and the particular systems.

OLIVIER DE WECK: OK, good. All right, so this is a really, really big deal in practice. If you do something that's-- so for example, if you make a design, you modify the interface because you think, ah, it's better for me. I'm going to change this. And it's different from the ICD. And then there's a failure in your system.

And then you say you're going to blame the supplier and say, your box failed. My mission got screwed up because of your box. And then it turns out, ah, you actually implemented an interface that's in conflict with the ICD. The supplier will step back and say, that's not my problem. You didn't follow the interface definition, and just changing one pin or one polarity can lead to a system failure. So really, you have to pay a lot of attention to the details here.

OK, let's talk about system integration. So what is it? It's essentially the process of deliberately-- so let's say you've designed your system. The interfaces are clear. When you physically put together your system, you have to integrate the system. So that means different things. Physical assembly of the parts, that's the liaison diagram, what fits into what. And we'll talk about sequencing in a minute.

You need to connect your different conduits and hoses. You need to fill in various consumables-- fuel, lubricants. An example in space flight would be if you're going to use a cryocooler, you might have to use some gases or liquids for the cyrocooler that are consumables, finite consumables.

Connecting all your electronics to the power sources, the avionics. Wire harnesses are still

used a lot. We would love to get rid of wire harnesses, right? They're heavy. They're cumbersome. They're expensive.

And we've been trying to replace wire harnesses with wireless, but it's really hard to make it work reliably. So we still use wire harnesses quite a bit. And then you need to upload and test your operational software that sits inside your system.

So that's a lot of work, and it has to be done very carefully. A lot of this is done in facilities that are very clean, clean rooms, because you really need to do every step properly. You need to document every step. There's quality control. And that's why it's expensive and slow.

So the next point here is the sequence in which the integration occurs may be important. And I want to-- so the reading, the post-reading associated with this lecture, is a paper by Ben-Asher, who's a professor at Technion in Israel who I think has done some of the best work, theoretical work, on system integration. And unfortunately, in complex systems, a lot of problems are errors. We often discover them during system integration and testing. So that's also a key point here.

OK, so let me-- I hope you have a chance to read this Ben-Asher paper, because it's pretty interesting. But I'll give you sort of the summary here on one slide. So what they looked at was, does it matter whether you do a single-stage integration, everything at once, or you do what's called incremental integration? So the idea of incremental integration is you just integrate two components or subsystems in isolation at a time. You check them, and then you gradually integrate as opposed to doing it all at once.

And the example in the paper is UAV, Unmanned Aerial Vehicle. And there's not a lot of detail, but there's four major subsystems-- the engine, the flight control system, the optronic payload, and then the payload control system. And so in one case, you would do a single-stage integration. So here's your four subsystems, and you just integrate them all at once. And that's shown here in this upper graph. So this is not the graph of components. This is the graph of integration steps.

And then you have to put the system, once it's integrated, these step seven, eight, nine, through different tests. And then the idea is that the amount of time that it takes to do each of these integration steps is somewhat stochastic. It's not a deterministic number. And so this upper graph here that looks kind of like a ski jump, this is time, number of days, versus the probability.

So what you see here is that it's right-skewed-- or left-skewed, depending on how you define it. But the peak here is to the right. And so the idea is that if everything goes super smoothly, actually, single-stage integration could work very well. But the likelihood of that happening is not high. And if you think about it, if you do all at once, single-stage integration, troubleshooting becomes more complex as well. And so that's why this curve is actually shifted to the right.

Incremental integration, in this case, you're just integrating two and two, two and two, and then you integrate your pre-integrated subsystems together. So it looks more gradual. And then here is-- again, using their model. They have made certain assumptions about this-- you get a somewhat broader distribution of outcomes. But the mean, the expected amount of time for integration, is actually less. So you have a faster integration time on average with incremental approach.

But in the worst case, you may be a little bit longer than the single-stage integration. But what really matters is, what's the average? So the conclusion in the paper is that the integration sequence is important, that the integration sequence can be optimized, that there's advantages of doing incremental integration, primarily from a risk perspective.

So again, Octanus 1 project, maybe you guys haven't thought about this too much yet. Maybe you have. But when you do your final vehicle, the sequence in which everything gets integrated and tested is important. That's really what this says. And an incremental approach can be very helpful.

The more components you have, the more the combinatorial space of integration sequences gets bigger and bigger. So you have to do this at some level of abstraction. Is that clear?

AUDIENCE: [INAUDIBLE]

OLIVIER DE WECK: Yeah. And so the way you describe-- this is driven by your supply chain. When are things becoming available? But you have to be careful because it could be sort of risk-driven as well. So OK, good. Any questions at MIT? Any comments about this?

AUDIENCE: I have a question.

OLIVIER DE WECK: Yup.

AUDIENCE: So I guess the takeaway is that incremental integration is usually preferred for most things, on average?

OLIVIER DE WECK: Yeah, I'm not sure. Read the paper. I think the conclusion is that for new systems that you've never integrated before, where your level of knowledge and the TRL levels-- or you've never done this before, first of a kind of integration, that incremental is better because if there's a problem, then troubleshooting is easier.

Now, if you're integrating a very well known system-- so it's sort of the 10th system, but you've sort of figured out where the bugs are-- so the risk level, the uncertainty is low. Maybe then, single-stage integration is OK. So it really depends on the novelty and risk.

So I don't think the conclusion of the paper is single-stage integration is bad, always. I think it's that you should think carefully about the sequencing, and it's driven by the amount of risk.

AUDIENCE: OK, thanks.

OLIVIER DE WECK: All right, good. So we're almost at the end here. I want to ask you a quick question. This is our concept question for today. And this is about interface standards.

So over time in different industries, rather than reinventing the wheel and coming up with custom interfaces for every project you do, you say, do the interface according to this standard or that standard. So I'm just curious to know from you guys-- and I asked this question last week from another class I teach at MIT. So please answer this, and then I'm going to show your answers and the answers of the other class combined.

So the interface definitions here are 802.11, [? MIL-STD-1553, ?] RS-232, BB_aJ23100. And then if you know of any other interface standards that you think are important, just click Other. This is a multi-- there's not just one right answer here. This is a multi-checkbox question.

All right, so here's the responses. So 56%-- so this is your results, and then my class from last week, kind of mixed together. So 56% know the 802.11. So what is-- I thought it might be more than 56%, but what is 802.11?

AUDIENCE: Wi-Fi.

OLIVIER DE WECK: It's the Wi-Fi standard, right? And now, G I think is the latest, I think. It's sort of ABC, 802.11 ABC. So it's kind of evolving standard-- very important, obviously.

[? MIL-STD-1553, ?] 24% know about that one. Anybody here, any [INAUDIBLE], [? MIL-STD-1553? ?] So you've done rural engineering, I can tell. So what is the 1553?

AUDIENCE: It's [INAUDIBLE].

OLIVIER DE WECK: Yes, it's a DoD. So that's what's a military standard. It's essentially a data bus. So for example, on military planes, on many satellites, it's essentially the architecture of the avionics. It's a bus architecture.

So you typically have like a mission computer that's the brain, and then it directs how the messages are sent from the different-- like the flight control system or the payload. It's kind of an old standard, but it's a very robust standard for avionics architectures.

OK, at MIT, RS-232? This is also kind of an older standard. Anybody chose that?

AUDIENCE: It's a--

OLIVIER DE WECK: Did anybody-- go ahead.

AUDIENCE: --communication protocol for electronics.

OLIVIER DE WECK: Right, but it's a very specific one. There's a keyword I'm looking for.

AUDIENCE: Serial.

OLIVIER DE WECK: Serial, right. It's a serial interface, as opposed to a parallel interface. So it's a serial interface. And still, it's kind of old-fashioned, but it's still used.

All right, now comes BB_aJ23100. And nobody, out of 119, knew what that is. So-- and this is kind of a trick question-- this is a BioBricks interface. So are you familiar with synthetic biology? So synthetic biology, the basic idea is that you can do what we do here for biological systems. So there's libraries of parts, standard library of biological parts, like different proteins, that you can actually assemble organisms.

There's an annual competition. It's been going on for almost a decade now. It's kind of scary, actually. But so far, nothing really bad has happened, I guess. But the idea is that in biology, it's going this way. So you can look this up. You can google for this particular reference, and

it'll guide you to essentially a protein interface. It's essentially a molecular, biological interface.

So the prediction here is that 10, 20, 30 years from now, people will know what this is. And there will be a very internationally recognized set of biological interfaces that people actually use and know. And then anybody chose Other here? Anybody choose other? Yeah, you did?

AUDIENCE: Like standard USB and HDMI?

OLIVIER DE WECK: And USB has evolved as well, right? We had USB 1.0, and then USB 2.0. Isn't that the latest still?

AUDIENCE: USB-C.

OLIVIER DE WECK: C, OK.

AUDIENCE: It's reversible.

OLIVIER DE WECK: Yeah, so standards evolve as well, right, as the technology behind the standards-- you know. So this is a huge deal in practice. Anybody at MIT chose Other, other examples?

AUDIENCE: Yeah, I chose Other. Things that come to mind are like SPI and I2C, more like serial electronic communication protocols that are useful for talking to chips.

OLIVIER DE WECK: OK, good. So those are sort of inside the IC world, right? OK, excellent. So all right, well, we are actually at the end of today's lecture. So I just want to summarize the key points.

Why is interface management important? There are many reasons, but I think the two top reasons are-- think of interfaces as a potential Achilles heel. They're potentially seeds for failures. We saw bottlenecks. We saw structural failures. We saw the wrong kind of information being passed through software. And so be careful.

And if there's nobody in charge in your project to really look after the interfaces, that's a problem. So you want to really think about this because almost any project today, you're decomposing it, and you're not doing everything in-house yourself. You're going to deal with suppliers. You're going to deal with partners.

So of course you have an organizational, contractual interface with them, but there's a physical interface as well. The longer you wait to define that interface with your partners and

suppliers, the fuzzier it is, the more problems you will have. So critical to really define the interface, and then stick with the interface very sharply.

Interface management has different aspects to it. So first of all, types of interfaces, there are many, many dozens or hundreds of types of interfaces, but they can all be boiled down to the four canonical types-- physical connection, mass, energy, and information flows. I've introduced you to the DSM, the Design Structure Matrix method. And that's where you basically decompose your system, and you show it as an n by n matrix. And then you put all the interfaces into that matrix that you know about, not just the interfaces that you want to have, but the interfaces that are actually there.

So if there's noise, or vibration, or EMI, Electromagnetic Interference, or waste heat, those are also important because you're going to have to deal with them in some way. Finally, ICDs. So ICDS-- and we said there's IRDs, Interface Requirements Documents, which are the requirements for the interface; IDDs, one-sided; and then ICDs are the most important. It's the two-sided interface definition. That's the NASA approach, but [? ISA ?] uses that. Many, many industries now use these interface control documents.

Ideally, in the long-term, we want to get rid of them. We don't want documents in system engineering. We just want to have models that we share. But it's still moving in that direction.

OK, and then finally, the point about system integration-- do not just do it randomly. Think about, how will you assemble the system physically? How are you going to connect all your electronics, the consumables?

How will you load your software? How will you test it? We'll talk about testing, verification and validation next week, but the way in which you integrate, the sequence of integration could be very important. And it's a pretty active area of research, so in sort of system engineering research, is really understanding and modeling system integration as something you can optimize.

And then the last point was industry standards. Really understand what are the key standards in your industry, what are the key interface standards. And if you're going to deviate, if you're going to choose something else than an industry standard, you better know why you're doing it. Because the minute you decide not to use an industry standard, you've probably just inflated the cost by a factor of 10 because now there's no longer any off-the-shelf components. You have to basically do everything yourself, but it's possible that there's good

reasons for doing that.