MIT CSAIL Alliances | Daniel_Jackson_Project_8(2)

Welcome to MIT's Computer Science and Artificial Intelligence Labs Alliances podcast. I'm Kara Miller. On today's show, software has become the foundation, the core, the currency of many companies. But when it comes to designing that software, sometimes, folks in charge shrug the shoulders, which can be a problem.

The companies that had leaders who were deeply invested in design, people like Steve Jobs at Apple, are the ones who were able to have the huge successes. And companies that generally don't pay a lot of attention to design at the leadership level end up with bad software.

Daniel Jackson, a professor of computer science at MIT and the author of*The Essence of Software,* argues if your design is flawed, so is your product.

Eames, the furniture designer, once said, the details are not the details. The details are the design. And if the details aren't good, then the whole design, essentially, can fall apart. So today, a discussion about why design is an ancillary, it's everything. That's coming right up.

In mid-March 2020, when much of the world seemed to shut down and people increasingly turned to software to solve their problems, one of the first things they needed was a virtual way to meet.

Back at the time of the pandemic, say, a few months prior to the pandemic and nobody had heard of Zoom, Everybody was using Skype and thought that Skype had a very bright future. And then Zoom came along, and basically, in the space of a couple of months, completely killed Skype.

So asks, Daniel Jackson, why did that happen? His answer is design. Getting someone to join your Skype meeting, well, it just wasn't as easy as it could have been.

In Skype, you had to-- if you wanted to have multiple people at a meeting, you had to create a group. Everybody had to have an account and had to sign on and so on. Zoom had this extraordinary idea, very simple idea of creating a link to a meeting that could be sent around asynchronously by email or whatever.

And people can then join-- and people could join who didn't have accounts. This idea, actually, wasn't invented by Zoom. But as far as I know, Zoom was the first company to really capitalize on this.

Skype may have missed a key design feature, but it wasn't because the product had been quickly thrown together. By the time the pandemic hit, it was 17 years old, and Microsoft had purchased it for more than $8 billion. Zoom was considerably younger.

It had been started by a former employee of Cisco who had new ideas about video conferencing, and he didn't really feel like he was being listened to at Cisco. When the pandemic hit, everything, of course, changed for Zoom, which, very quickly, seemed to eclipse Skype.

And this was a huge win. And I think it's a very nice demonstration of the idea that if you have one really good concept, you can dominate the market, and you can make your product different.

That is a story of design, giving you an edge. But Jackson says, even Zoom has some aspects of its design that could use a little improving, features that can maybe slow things down a bit, gum up the works, and generally make the user feel a little less awesome about the product.

One of them is this funny thing that happens with so-called reactions, things like raising your hand and so on. And the truth is, it's all a bit of a mess. And many of us know it's a mess because we've been in so many meetings where people say, hey, Kara, is your hand still up, or did you leave it up from the last time. And that's evidence that this feature, this concept, as I'd call it, is really messed up.

But that's not Jackson's only complaint.

I actually think there's a much bigger design problem in Zoom. And if the C-suite of Zoom and designers in Zoom are listening, I would love them to engage with me on this question, because I think it boggles my mind that they're not addressing this, which is that Zoom, essentially, has no way for you to track your upcoming meetings.

And we all have loads of upcoming Zoom meetings. They have a calendar integration that is really clunky and doesn't work very well. And they have this panel that shows scheduled meetings, but those are only meetings that you've created and you hosted. So if someone else creates a Zoom meeting, and they send you the URL of that meeting, there's no way for that to be, somehow, present in your Zoom app.

And there's no way for you to see a simple list of upcoming Zoom meetings. It's so bad, that if you're in a Zoom meeting and the Zoom meeting dies, Zoom doesn't even remember what meeting you were in. So you can't even go back to the app and say, reopen, rejoin that meeting I was just in.

Jackson says design features like this can be serious business, and he's pretty sure Microsoft and Google know that it would be to their advantage to create a product without the friction, and they're working on it. After all, consider why Zoom dethroned Skype in the first place.

Rather minor design feature that turned out to be not so minor. Zoom could be offering potentially a calendar of all the meetings you're in with the ones that you're hosting, let's say, shaded in a different color, all the URLs in one place. Easy and distilled.

I'm not saying that this is a trivial piece of design. It would need to be designed very carefully. What astonishes me more is that it's just completely missing, and that Zoom appears to be focusing on the cream on the cake, things like recognizing hand gestures, which, I think, for many people are cute, but really doesn't bring much value. And they're somehow missing an opportunity to ensure that their product really works smoothly in the context of how people actually live their lives and use Zoom.

Jackson argues that this is one example of the kinds of design improvements that could be made much more broadly, the kinds of improvements that, frankly, can make a product. And that's a very Steve Jobs way of looking at the world. But finding the right people to make those improvements, it can be tougher than you think.

One of the big paradoxes is that in most companies, there's nobody called a software designer. Mitch Kapor actually gave the very famous talk to a collection of CEOs back in 1996, brought together by Esther Dyson. And his talk became what was called a manifesto for software design.

And in that talk, he pointed out that there's no one analogous to the architect of a building in most software organizations. And now here we are, more than 20 years later, and we're still in that same situation, actually. We never did what Mitch Kapor wanted, which was to create a role for a software designer.

And instead, we've siloed software design into multiple different roles. So we have UX designers. We have product managers. We have software architects. We have strategists. We have marketing people. And they all do little pieces of software design, and they often don't work in a very coherent way together. And the result is that software design is often not done in a very systematic and coherent way.

So if you're a CEO or a CTO, your EE don't talk to people who design software, maybe, all the time, or you don't think about the fine points of it all the time. Why should you care that, as you say, it's cut up into these little chunks and distribute it, not, as you say, one person's job, but maybe a lot of people have a hand in it.

Sure. Well, first of all, if your company's business is producing software, you want that software to be good. And if you don't design it intentionally, it won't be good. And if you have lots of people playing different roles, then, typically, what will happen is that crucial parts of the design will fall between roles.

And so you might end up with something that looks like and has some basic good functionality, but it doesn't all fit together properly, because there was no one who was responsible for actually making everything coherent and making the different roles fit together.

And if your business isn't actually building software, but you need software in your company, then if you have bad software, you're not going to be able to run your company well. So I think having a grip on how software is designed in your company is really essential.

And as we've seen, the companies that had leaders who were deeply invested in design, people like Steve Jobs at Apple, are the ones who were able to have the huge successes. And companies that generally don't pay a lot of attention to design at the leadership level end up with bad software.

You talked about if your company is about software, design matters. I think there's no question that in the last 10, 20, 30 years, many, many more companies have become, if not purely about software, largely about-- I think about the Marc Andreessen quote, "Software is eating the world." Software is really important. And when you say that if you don't really focus on this, you're going to design bad software, is that happening a lot, that people are designing bad software?

Well, I think designing bad software suggests that you're trying really hard to do it, and it's not coming out so well. What happens more often is that you don't design is that the qualities of your software evolve by accident. Often, for example, programmers, software engineers will fill in what seemed to be minor details.

But as Eames, the furniture designer, once said, the details are not the details. The details are the design. And if the details aren't good, then the whole design, essentially, can fall apart. And so that's what often happens. It's not that people who really focus on design don't do a good job. It's more that people pay lip service to design, and then it never really happens.

So if somebody in leadership came to you and they probably do, and they said, OK, all right. So you're saying I've been maybe going about this wrong. We don't think about this much. It is like whatever the details will come together. We hope they will. How should I be thinking about things differently?

Well, first of all, I think that there's a mindset shift that would be helpful for a lot of leaders in companies. The first is to recognize that complexity exacts a much higher cost than people imagine. We even have this name in software development of technical debt. And the idea is, somehow, that you can go into debt and pay off the debt later.

But it's really a terrible analogy. Because the truth is that, unlike a normal financial debt, in which you can invest the money and make profits and then pay the debt back, the low interest rate, the interest rate is extortionate for software. And when you've actually introduced the complexity at the beginning, you basically can't recover from it, normally.

So that's the first thing I'd say, that you need to actually be able to assess the costs of complexity. The second thing is, and it's related to this, you need to understand the implications of friction for users. Very small amounts of friction can basically render a system nearly unusable to a user, or at least make them not want to use the system. And if you're trying to sell a product, that can be debilitating.

So I think it's really important for leadership and companies to be able to focus on what is needed to make a simple and compelling product that will convey a very clear understanding of what the software does and how it does. It will be easy for users to be able to grasp. And that means going beyond the normal superficial qualities that people often focus on, things like the look of the user interface, and all that kind of stuff.

And it's interesting when you talk about technical debt, this idea of like, well, we'll just do this, and we'll go back in later. You talk to software engineers. Nobody wants to be the person who then, later, has to go. It is like an act of drudgery, to go in, to try to fix it. Nobody's happy that they have to be the person to do that.

That's right. It's too painful, and it's painful for many reasons. It's painful--

You don't really want a job to be painful.

That's right. That's right. Nobody wants to do it. It's much more-- it's much more fun to do good design up front. But design up front has been devalued over the years. The agilists, the people promoting what was called agile development actually coined a disparaging acronym. They called it BDUF, big design upfront.

And there was some rationale for this, which was that people were often producing big design documents and requirements documents that weren't that helpful. But this also devalued the whole idea of actually getting your design straight at the beginning. And if you don't do that, you indeed pay a very heavy price because you end up with all the different parts of your software, coupled together.

So you can't make a change in one place. You need to make it in lots of different places. And it's a very dispiriting challenge to have to face, to take a mass of complexity, which, by the way, your users have learned, so they're going to have to also unlearn the way they use the software. So that's a big hurdle for many companies. And to roll things back, it's often almost an impossibility.

As you move through the world and encounter different software, do you feel like you see a lot of stuff that you think, this is not good. This could be so much better.

Well, it's a funny thing. So in my book, I focused on what you might call personal productivity software, the kind of software produced by companies like Google and Apple and Microsoft and Dropbox and Adobe and so on. And I focused on that software because it's the very best, and because I didn't want to be dealing with strawmen.

I wanted to show that the ideas I had could apply, not only to software that was egregiously bad, but to really good software. And so I spent a lot of time focusing on those products and essentially trying to identify their flaws and to find ways in which they could work better. But the truth is, most of those products are really great.

And on the other hand, I'm not sure that those are the products that people spend most of their life using. I think if you look, for example, at enterprise software, the kind of software built with products like SAP and Salesforce, which, I should say, I think are a great products, the problem is the software that's built with them is often really poor. I mean, just look at the software that we have to deal with at MIT, all the internal software. You look at systems like--

You think MIT might have a corner on the really great software market. But I guess not.

It's all built by the same way, which is that these systems are typically one off systems, right? They're built for a customer, often by a consulting company that has no motivation, no market pressure to do a particularly good job. They just have to make it work well enough that organization or enterprise can use it.

And the result is a huge gap between the quality of that enterprise software that many of us have to use in our daily jobs and the kind of software that is out there in the world produced by the Microsofts and Apples of the world. And it's an egregious and huge gap. And it's a strange phenomenon. But I think it's due to market forces that, that gap hasn't been narrowed.

So you talked about enterprise software, like once you move out of some of these big companies, things can be a lot more touch and go. But you also mentioned Steve Jobs. I guess I wonder, given the success of jobs and even, in fact, when you think about the jobs Wozniak partnership Jobs was really not the technical person.

He got so much attention, mostly because he cared about how things are designed. How did we get here, where so many companies think of design as an afterthought?

I'm not sure they do think of it as an afterthought. I think most companies actually realize that design is central to their business. They're just not sure how to do it. They might realize, when a user interface isn't very good and, to some extent, that problem I think is nailed, right? We know how to design good user interfaces now.

We've had decades of human-centered design, started by people like Don Norman. We know how to make a good user interface. And I think most companies have fixed that problem. What they, I think, need help with is getting to what Jobs referred to as the soul of the machine. How do you actually make the underlying concepts of the software simple, flexible, powerful, intelligible to the users?

And that's a much harder problem, and it needs a different investment. And I think a lot of companies, they get a little bewitched by technology. They want to jump on the crypto bandwagon or the AI bandwagon or whatever, and they're less willing to really invest in making their software design so that it's really, truly usable.

OK, So in that spirit of usability, let's dive into another specific example of a product where, again, the software is pretty much everything. We talked about Zoom before. Let's look at Dropbox now. It's very widely used. I've certainly used Dropbox a million times. What I did not realize is the intricacies of the way it works, which you which you talk about in the book.

And you had the experience of having a friend. She was running out of space on her laptop. Do you want to take it from there? Certainly, everybody has been in this situation, right? You have to figure out what you've got to delete, what's important, what's not important anymore.

This was actually my daughter, and she allowed me to mention that it was her. She was she was on her laptop one day, and she was running out of space. So very cleverly, there's a feature on a Mac. You can say, show me all the files bigger than 100 megabytes or whatever.

And it produced a couple of gigabyte-sized files, and she didn't recognize any of the names. So she just deleted them. Now, she didn't actually delete them permanently. She just put them in her trash. Well, a few minutes later, she started getting frantic calls from her workmates, saying what happened to all our data?

And then she eventually realized that what she'd actually done was to take a bunch of very big data files that were in a shared Dropbox folder. And by moving them to her trash, she'd moved them out of the Dropbox folder on her Mac, and that had essentially propagated a deletion to the shared Dropbox folder that everybody else had. So it essentially caused the data files to disappear.

So if you and I share Dropbox files, and I delete them, I've also deleted stuff that you have?

Well, that's actually a complicated question.

Because I would assume, no. I would assume it's on Daniel's computer, not on my computer.

So let me first say before I bash Dropbox too much. I think Dropbox is a really great product. It's very well designed. And as I explain in my book here, to some extent, Dropbox has inherited a subtle design flaw from the Unix operating system. And for those of you who are geeks, you can read my book. And there's a long piece explaining all the details here.

In short, it's actually worse than this. And the problem is that whether or not deleting a file from your Dropbox causes it to be deleted from the owner's Dropbox depends on exactly what the context is. So, for example, if you have a big file and you just share that file with me, and I delete that file, then it won't actually be deleted from your Dropbox.

But if you share a folder that contains that file and I delete that file within the folder, then it will be deleted. And in my book, I explained why this is the case. And it's actually quite straightforward to explain, but it's completely confusing to most users.

Well, you would hope that, for the users, things are simple and obvious. But when you say like, well, the difference is a file versus a folder, nobody's going to remember that distinction. And when they go to delete things, and then everybody's going to be in a heap of trouble.

Yeah, that's right. I mean, it was very funny, actually. So when I wrote the book, I was a little bit nervous of what I was saying about Dropbox was so obvious to computer scientists that they'd laugh at me, and they would say, you really feel you need to explain this to people?

And so I ran that chapter by a couple of computer scientists friends, and said, what do you think? Is this your experience of Dropbox? And do you think this is a reasonable explanation? And to a person, what a whole bunch of senior computer scientists said to me was, oh my goodness, I would never dream of even trying to delete something in Dropbox because I have no idea how it behaves.

And I thought that was extremely revealing, that they basically knew enough to just not to even try that. But I think that's a problem, because I think it means that there's a lack of confidence in understanding how the software operates. And that means that if you can't use your software with confidence, you can't predict what it's going to do, then things are difficult.

Rodney Brooks, who has long been here at CSAIL, a pioneering roboticist, once told me the story of he sent some robots into the field for the military. And the people he was working with, his colleagues were complaining because they were like these people in the military, they don't know how to use these, what's wrong.

And he was like, no, no, no, no. It's our problem. If they don't know how to use these things, that means we have not done a good job designing them. Because our job is to design these robots so that people with no specific expertise can use them.

I think that's generally exactly right. And one of the things that I describe in my book is this idea of how you can build software out of familiar concepts. And if you build it out of familiar concepts, then there's nothing for the user to learn. Now that said, certain kinds of specialized software, your user is going to need to educate themselves.

Let's take, for example, Photoshop. If you want to be able to be a skillful user of Photoshop, you've got to figure out what a layer and a mask are. And that's not something trivial. It requires a bit of education. I eventually realized this myself. You can't watch endless YouTube videos on how to remove red eye. They won't help you.

You can't just mimic the tutorial. At some point, you need to understand the concept of a layer, and it's a subtle concept and a very powerful one. So I would say, Brooks was right, in general, but there are exceptions to that.

Let me just come back to your daughter. So she put a folder in the trash, a Dropbox folder, right?

Actually, just individual files.

Individual files in the trash. And then everybody started freaking out, who worked with her, and what happened?

Well, actually, it turns out that it wasn't a disaster after all because Dropbox has a very nice concept-- originally invented by apple-- which is the concept of the trash. And so what actually happens is when somebody deletes something from their Dropbox, it gets moved to the trash.

And so we were able to go to her trash and reinstate those files. And so disaster was averted. And Dropbox has a number of concepts that essentially make accidental deletions much less likely. One is, for example, just the whole concept of access control, that you can actually ensure that only certain users have the ability to write or delete certain files.

So in the end, it wasn't so bad. But there have been cases of people losing all kinds of stuff. My book describes a case in which some poor Google user lost basically their entire life's collection of files from their Google Drive, and they were so angry about it. They created a domain name-- which I won't actually say on the air because it was obscene.

But they created a domain name to register their frustration with Google Drive and the way that it lost their files. So this is actually a particularly intricate design problem that Dropbox is facing here. And it's a similar design problem faced by Drive and Box and all these other products as well.

Have you heard from Dropbox since the book came out? Do you have any sense that they're working on this or--

No, I'd love to-- I'd love to talk to them about it. We've had some wonderful engagements with Drew Houston and his team. They've actually been very generous funders of a fantastic new program at MIT for AI-augmented work. And so we're in the process of having very interesting discussions with them.

And they have a very interesting and profound commitment to the use of AI in software. So there's lots of exciting things going on at Dropbox. But this one thing I haven't had a chance to discuss with them, but I think it's an interesting one. I'd love that opportunity. So if someone is listening--

Yeah, yeah, exactly. When you talk to people, both who run companies, but also people who are in the trenches designing software-- I mentioned the Rodney Brooks thing-- what do you think they get wrong about ordinary people and how they function versus-- do you know what I mean-- versus the people on the inside and maybe the more technical people, how those two different groups of people think.

Well, I would say that, we all get these things wrong, first of all, so I don't want to be disparaging a lot of great designers out there. But I think one of the things that we're all easily misled by is the idea that when we have a really cool idea that our users want to learn it, they don't.

People don't want to learn new stuff, unless they really have to. And so one of the key ideas in my book, as I mentioned before, is the idea that, often, you can achieve new functionality by using old familiar concepts. I'll just give you one example of that. Keynote, Apple's slide presentation software, has this really beautiful scheme in which you can basically take a slide, and you can nudge it right in the Navigator view.

And when you nudge a slide right, it makes it a child of the previous slide. And this allows you to create a hierarchy of slides and to open and close groups of slides together. And when you see this, you understand immediately how it works, because it's just the old outliner concept, where you could have different levels of an outline.

Now if you look at Microsoft PowerPoint instead, Powerpoint, trying to emulate this same functionality, has a concept of what they call sections, but they invented a new concept. And to be honest, it's completely obscure and baroque, and it's really hard to figure out, what creating a new section does. It's also, by the way, less flexible.

You can't have sections inside sections. You can only have one level of sections. And so they essentially invented a new concept that was less powerful than one they might have adopted. So I think that's the first thing we need to know about users, which is that they really want things that are old and familiar to them.

I think the second thing is that we underestimate the extent to which a tiny little bit of friction can put a user off and the extent to which a user can sour on a product because they don't really understand the product properly. I mean, I'll give you one, one simple example. I just went traveling, and I used this fantastic eSIM product from a company called Airalo.

And it's really great. It allows you essentially to switch SIM cards, but without actually putting a physical SIM card in. You just download this thing called an eSIM, and it's a beautiful idea. And the software is great, and the price is great. And it works really well. The problem is that there are a couple of little bits of friction in the experience, which are really maddening.

And so, for example, when you put an eSIM in, you can attach a label to it that says, this is my eSIM for Spain, if I'm visiting Spain or whatever. So that when you look at your cellular options, you see one that says Spain. But somehow, this ability to attach a label is just not clear.

This concept is not clear in the design of Airalo. And if you miss the opportunity to create the label at the right point, you can't basically do it later. And it makes the experience just a little clunky. And I think damages, and otherwise, really excellent product.

Yeah, and people get frustrated easily.

People get frustrated very easily. You know, when you talk before about Microsoft and this fix or augmentation that they made to PowerPoint, and that it's clunky, and I think you said baroque. I mean, Microsoft isn't like a fly by night organization. How does that get through the gatekeepers? And a bunch of people say, yeah, that looks great. Let's go with that.

Well, I think, to be fair, these products are amalgamations of large numbers of concepts. And they're often created in a distributed fashion, with different product managers having responsible for different parts. And they might not be understood to be, somehow, central to the product.

It wouldn't surprise me if most users of PowerPoint don't use the section concept. Now when I think becomes a problem is when you have a concept that turns out to be completely central to the product. And in that case, I think, often, the development teams know that they have an issue.

So just to go back to another Microsoft example. If we consider Teams, for example, Teams for a long time, I found, at least, it was almost impossible to join a Teams meeting. And it was something to do with the whole way account management worked.

And I know from talking to colleagues at Microsoft that this concept of what it meant to be a user of Teams and what it meant to belong to a team was essentially a concept that had evolved in a confusing way and that amalgamated different functionality for different kinds of user databases and so on.

But the result is that the simple act of joining a meeting becomes difficult. Slack is another example of this, by the way. In fact, I have a suspicion that this whole question of managing accounts, particularly when accounts are associated with institutions, it has become quite problematic. So in Slack, for example, you have your personal account.

And I have my MIT account, and then you have your workspaces and your channels and da, da, da, da, da. And it's really almost impossible when you want to join a new Slack workspace to figure out what the heck is going on. Because the concepts, to be honest, are such a mess.

Let's talk a little bit about AI. We've certainly talked about it a bunch on this show. And I know you've spoken with computer scientists who think AI is going to reshape this field. So what's your feeling about how AI will affect software design. Because AI can, unquestionably, be helpful. But I'm not sure that it can tell you what it is you're aiming for, and that's something that, as you write about, people are not actually always clear on.

That's exactly right. It's funny, when this whole, big GPT thing came out, I like a lot of other people who was not working in this area. I thought, no, that's the end of my career. That's the end of anyone being interested in software design or software development or software engineering.

It's all just AI and chatbots and LLMs now. There's nothing left. And to some extent, that field has been sucking a lot of oxygen out of the air. But then I came to realize that, actually, this was really good news for my field. And the reason, in short, was that LLMs make important all the things that I think really have mattered.

I've been telling our students-- we've been telling our students for years, you might think that what you really need to do is to learn every intricacy of the programming language and APIs, become a great programmer, but really, thinking straight is more important. And being able to think about high-level design and conceptual design, that matters much more.

And the truth is, it was always a hard sell. But it's become a lot easier. And now that LLMs can do all that programming, because it makes it very clear to students that they have a lot of competition there, from machines who are a lot cheaper than they are. So I think that AI is producing a huge opportunity to, essentially, reinvigorate a lot of the fundamental ideas in software design and software engineering.

And I think we're going to have to change a lot of our fundamental assumptions. Over the last few years, an agile perspective was becoming dominant in which people said, forget all that stuff about specification and design. It's just the code. Just write the code. Because the code is the product.

The code is what delivers value. And that philosophy is looking pretty thin now, right? Because, now, the code is not the value. The code is what the LLM can produce for you, and the real value is in the specification and design work. So I think agile is on its last legs.

And do you think LLMs will revolutionize things because, in some ways, it helps people get across the bridge. So much more quickly. And really now, the goal of the designer is to think, well, where do I want this bridge to-- I can build the bridge fast, but where do I want it to go?

I think that's exactly right. We're having some very fun experiments that we're doing. I have a team, and we're basically building an application builder that allows you to describe the fundamental concepts of the application and then uses GPT to generate the implementations of those concepts and to merge them together into an entire application.

And to be honest, it just works beautifully. And it makes me wonder, in a few year's time, how many people are really going to be writing code, at least, for many of the kind of applications.

I was going to ask you that, is it going to be fewer people?

For sure.

So it sounds like it pushes people up a little bit to more of a conceptual role.

I think that's right. And I think that's very exciting.

And less of like, just crank out that code role.

Exactly. Yeah.

So we talked about that. There's the other piece of what AI could do, which is AI could potentially, in theory, help me understand. If I do this thing in Dropbox, am I going to delete all the stuff I shared with Daniel? I have a bunch of stuff shared with all these people at work or whatever, if I do this, what will happen? Are the LLMs up to answering those questions to being my helper, to being my computer scientist on the side?

Well, I think LLMs, obviously, they're up to the task. But an LLM can only answer questions accurately if it was trained on data that is sufficient to answer that question. And so if you look at that particular question of, if I delete a file in Dropbox, will it disappear also from the owner?

I've actually done some experiments, using GPT to see how well it can answer that question. And it can't answer that question reliably. Sometimes, it gives one answer. Sometimes, it gives another answer, depending exactly on the context. And this is merely a reflection of the lack of conceptual clarity in all the material that's available online.

And you can read a lot of stuff where people try and explain how Dropbox works, and they and get it wrong. And I think the reason they get it wrong and, subsequently, the reason that GPT is confused is because the underlying concept is just too complicated.

So finally, step back for a second. Software design is something you really care about. Do you feel hopeful about the direction of software design? I don't know. What do you see ahead? Because I also feel like you do talk about this missing piece of people have to know what it is they're aiming for. They have to be thinking clearly for all of this stuff to come together. And I don't know if there's enough of that.

I think there is. I'm very hopeful. I think we're in an incredibly exciting time. I think the companies that we've been talking about are full of extremely talented people. I think that one of the exciting things that happened is that we've essentially knocked off some of the less interesting problems.

I think, for example, user interfaces. We've nailed that. We've had so many years of really great work in designing user interfaces, and now we have all these so-called design systems to help you design your user interface. It's just a solve problem. And it can be done very straightforwardly.

And I'm sure LLMs are going to do a large part of that. And similarly, a lot of programming has now been essentially, can be delegated or will soon be able to be delegated to LLMs. And that leaves us the much more interesting tasks, as you say, of figuring out what are we trying to do?

And what are the essential details of how this product is going to work? And I think we'll be able to use LLMs to help us do that, but it's still going to mean that we're going to need to make the fundamental decisions ourselves and have the fundamental insights that will really distinguish great products from mediocre products.

So I see just boundless opportunity. I see amongst my students, a group of people, who are committed not only to producing great software, but also to producing software that does good in the world. And I see a lot of companies that are very committed to design and are doing great work. And I think it's a very bright future indeed.

And does that mean that more people who run companies have to care about how software is designed and think of it not as an afterthought, but as a really core, important thing?

Absolutely. I mean, software is your biggest asset, and it becomes a liability very quickly, if it's complicated and clunky.

Daniel Jackson is the author of *The Essence of Software.* He's a professor of computer science at MIT, is an associate director of CSAIL. Thanks so much for being here.

Thank you so much.

And before we go, if human-computer interaction is something that you're interested in learning more about, check out CSAIL's upcoming online course. It brings together eight CSAIL experts to go beyond user interface design to teach you how rapidly the field is evolving. Podcast listeners get a discount.

If you'd like more information, just email us. podcast@csail.mit.edu. And CSAIL is C-S-A-I-L. So podcast at podcast@csail.mit.edu. I'm Kara Miller. Our show is produced by Matt Purdy and Nate Caldwell, with help from Audrey Woods. Tune in next time for a brand new edition of the *CSAIL Alliances Podcast* and stay ahead of the curve.