What is a port? What is the difference between an interface and a port?

Guy Atkinson interviewing Alistair Cockburn, XTC London, May 21, 2024

(Audio recording on YouTube: https://www.youtube.com/watch?v=x6H2n_RAZoA

There you go. And we can close. Yeah, yeah, that's okay. No, no, that's

So my problem is that people draw hexagons everywhere. Hexagons don't exist in the code. They're not real. People draw lines. Lines don't exist in the code. They're not real.

So what I'm battling against is people just literally drawing hexagons everywhere and said I did it. And I go, no you didn't. And then I have to ask, of course, why, how do I know you didn't do it? What is it you didn't do?

Well, the first thing is they don't put tests there. If you don't have tests, you have no protection. So, a hexagon is really what we learned recently.

It's a component. In the UML sense, it's a component. Like, if you get a chip from a catalog, right, what are the characteristics of a chip from a catalog? It's got input pins, it's got output pins. You, the user, don't get to choose what the output pins output. And by the way, they test it in the factory.

So the two characteristics are, the chip owns the interface, and they have tests. So when I look at a piece of code and they say, they draw the line, I have to see that the *thing* owns the interface, and they have tests. Well now we get to the question, what is, this was the question, what is the difference between merely an interface and a port?

G: Which is a special kind of interface.

Which is something, an interface plus something. And the reason, the background is, that this colleague of mine, I have high respect for him, drew an architectural picture, wrote a blog entry, and he had where I would put the hexagon next to the technology, the real technology boundary. And he drew it inside. At the bounded context in DDD terms.

G: So inside the component?

Inside what *I* would call the component. He declared that the bounded context was a component. Or it was a hexagon. He said it's a hexagon. Hexagons have no meaning, but that's what he said. So the test question I asked him, I said, well, I foresee a problem that you have to write tests around your hexagon, and you also have to write tests actually at the technology boundary. You've got a team of people, I can't imagine they're going to maintain all of those tests.

And he said, no, they only maintain them at the technology boundary.

And I said, your inner thing is not a hexagon, in the sense I mean it. It's not a component. It doesn't have tests.

It doesn't. It's nice, but the line doesn't mean anything.

So then I had to face up to the question, if I look at code, how do I know if it is or it isn't? And so the test question,

G: Is or isn't?

Doesn't or doesn't implement the Ports and Adapter architecture, to use the long words very correctly. Right? Okay. Is or isn't a hexagon, loosely speaking, but really

implements the Ports and Adapter architecture. Right. That means it has to have Ports.

So now we're at the how many angels dance on the head of a pin question. That I was asking those people, what is the difference between an interface, merely an interface, and a port?

G: Which is also an interface.

Which is an interface plus something. What's the plus of So that was the test question I gave them, because I have to face it, it's a great question.

Nobody knows it because it's not a question that gets asked. Right? So I don't even have the perfect answer, like I'm noodling toward it.

But the present, but I used a chip analogy. Now there's two characteristics.

The chip manufacturer owns the interface. So if we look at, Java turns out to be perfect for expressing this, and we have a package diagram or something. In an ordinary, kind of a system, you have a component, I'll use the, the glass as the component, right? And, and, and, and this is a boundary.
And the other thing on the outside, declares an interface. And the writers of this wine glass,

G: This is the component, it's the That's the component, it's the wine glass.

The writers of this component of this wine glass, have to code to the interface of the wine bottle. Of the external thing. That's normal, right? If you get a database, if you get a database, you have to code to the interface the database provider told you.

G: Oh, okay. So we modify Because the database existed first.

Exactly. It came from the catalog. You can't change it. It's there.

G: Oh, okay.

Doesn't matter. HTTP, database, file system. Yeah. Anything, right? Yeah. You bring in a thing, it owns its interface. Yeah. You have to modify your code to meet that interface. This thing has an interface. That, that, the code. Calls it makes out are the outgoing interface. But these people have to meet the requirements of the, what's called provided interface of the, the thing on the outside.

G: So maybe SQL, for example.

For exactly, exactly. What's different, in order to make it like a chip, this thing has to own its own own interface.

G: The caller.

Yep. The caller, the wine glass in this case. Mhm. Declares the, the verbs that we'll use to call out.

G: Right. This is the subset of SQL I'm going to use.

I'm not even going to use SQL. I'm going to speak in my own language.
I don't know if SQL is there or not. There might be no SQL. There might be a flat file. There might be a human. There might be a UI.

G: I do whatever I want here.

I do. It's, it's very godlike. It's very haughty. I mean, it's really, really egotistical. Could be, yeah. It says I will only talk to the rest of the world. In this way, if you want to deal with me, you must honor my language.

G: Yes.

Right? That's important. That's the part of the port, like a chip. Right? So you make a component, like it could be in a component catalog.

G: Can I clarify the recording? We've got two components. One is the call of the wine glass, and the other is the standard thing from the library, which is, um, a chip, or the wine bottle, and, which could be a chip.
06:40 or a SQL database.

And now, guess what? Because each of these has declared it owns the vocabulary. Yes. And they're different.

G: Yes. Now I did not understand that back then.

So guess what? You have to put an adapter between them.

G: Adapter, right.

That's where the adapter comes from.

G: Gotcha.

This thing, your system, declared I will only speak in my terms, which is a domain driven term. There's no reference to technology. I, I will say what I want.

G: Business and domain driven.

And somebody, you have to integrate it with somebody else's thing where they did the same thing.
And so you need to write.

G: From a technology point of view, right.

Right, and so you have to write a translator, an adapter. Right? So the thing is, if, if, now let's go to a Java packaging diagram or a Java whatever diagram.

G: So more strongly typed.

Yep. Because Ruby, it's so soft, you know, this discussion almost can't exist, right? It almost doesn't exist in Ruby because there's no declarations of anything, right? But anyway, for Java, so normally, we would have the technology thing, the external thing, and we'd have that it declares an interface, and we'd have that this uses, with a dashed line, uses that interface.

G: The dashed line being without the adapter.

Without, what, what does Whatever, yes, it will, it will, the programmers here will adjust their code to meet whatever the external thing said, like SQL being an effective example, right?

Where was I going to go with this, one second, one second, let me just catch up.

Now what we're going to do, is we're going to say, move that interface, this interface We're going to move the interface definition inside the component, inside our thing, right?

So this thing now declares an interface. And UML calls that a required interface. Provided interface is the normal calling interface that we're used to.

G: Such as the SQL one.

Yep, yep. And provided is, in Ruby you can't even see it. It doesn't exist, right? It's just, we just make calls.

G: You could have a comment or something.

Yeah, yeah, yeah. But you wouldn't see it. You wouldn't see it. But in Java, in Java, you literally can say, this is the interface I'm going to use. They're the interface.

G: Right.

And then the user's relationship is internal to our little thingy.

G: Right.

And now, we have this thing, and they do this little vertical arrow, right, with the big triangle head there.
This implements that interface. This thing, well in this case, excuse me, our adapter. Adapter, okay. Implements this interface.

G: The required interface.

The required interface. Yes. And uses the provided interface of the, of the comp, I'll say SQL just to make it sound better.

G: *The of the SQL, right?*

That's why we have to have an adapter. Yes. But notice that our thing owns that interface.

G: The left-hand side of the interface.

Owns the, owns the declar, the definition of the interface. And anything that wants to talk to us has to do and implements that. So when you look at the package dependency, right? The external thing has a package dependency on our component. Yes.

Whereas if I didn't have the adapter and we were doing it the other way,
(recording ends)