The problem with open source software

Welcome back, …! It's Jess.
And today we talk about the biggest—or certainly one of the biggest problems with open-source software development.
We speak to the author, software developer, and teacher, Huw Collingbourne.
It all comes down to modularity.
People who come up through the open-source community, if you like, is that, while many people who get used to contributing a quick fix, a quick hack, a quick clever piece of code,
they often don't have the long-term involvement in that project to make sure that their code is really maintainable,
has no side effects on somebody else's code,
and maintains compatibility with all sorts of other programs that might form part of the project.
Two things for maintainable code are clarity and modularity.
By clarity I mean that your code should be readable by somebody else.
You should be able to give your code to somebody else, and they should be able to make sense of it.
Now, there's a lot of people—are able to do clever coding that is not particularly clear.
So it might do the job, and it might do it very effectively, and it might do it very efficiently,
and they might have thought of good ways of getting some particular task done,
but they give the code to somebody else,
or becomes part of a project—
which twelve months, or two years, or three years down the line,
somebody else, who's never even met the original programmer, doesn't know who wrote that code.
Somebody else is given that code, and they have to make it work.
And it has to be clear.
The other thing—and it's a related feature, really—is modularity.
Now, by modularity I mean that each piece of code that does a specific job should be in a clearly defined place,
and it shouldn't have side effects upon code that's outside that clearly defined space.
Now this I'll—I'll say a little more about modularity,
because a lot of programmers these days are not really completely familiar with that concept.
In object-oriented programming—… is an object-oriented language—a lot of people are familiar with the idea of encapsulation.
And so that means you put data, the variables and the behaviour—that's the methods that act on the data—inside the class and you create objects from them.
But they're not so familiar with the idea of modularity,
by which I mean there should be one way in to that data and one way out.
And when somebody recodes, for example, the behaviour—the implementation of a method,
that should not have any possibility of creating unexpected side effects on objects or on code outside the class in which it's defined.
And a lot of people find that quite an unusual idea.
I mean, I happened to come across this, really, I suppose through a different line of languages.
It's the Pascal line of languages, and in particular a language called Modula-2, that really really emphasized the importance of modularity.
And so I was exposed to those ideas many years ago, quite early on in my programming experience.
I've taken up with me throughout every other language that I've come across that idea of keeping everything absolutely clear and completely modular.
And I think it's a shame, really, that lots of people these days who are learning to program are unfamiliar with that idea,
so that—so they encapsulate data and behaviour but they don't pay any close attention to what I call modularity,
that is, the ability to isolate the behaviour from the programming world outside the module or outside the object in which it occurs.