Some Elements of *Mathematica* Design

Stephen Wolfram

"WHY DON'T YOU MAKE `ListPlot` work with complex numbers?" an experienced *Mathematica* user once asked me. "What should it do?" I responded. "Make a plot in the complex plane," he said. "Plot each point using the real and imaginary parts as x and y coordinates."

For a moment, it seemed like a pretty good idea. I had often needed to make such a plot myself. I had usually defined a simple function to do it:

```
ComplexListPlot[list_] :=
     ListPlot[Transpose[{Re[list], Im[list]}]].
```

Perhaps `ListPlot` should do this automatically.

But then I realized the problem. Say I did `ListPlot[{5, 6, 3+I}]`. I would get a plot with three points in the complex plane. The first two would be on the real axis. But what if I just did `ListPlot[{5, 6}]`? Something completely different would happen. Now `ListPlot` would do what it usually does and use 5 and 6 as y coordinates. What a mess!

So what, you may ask. Wouldn't anyone who was plotting a list know whether it was a list of complex numbers or not? Perhaps so if they had just typed in the list. But if the list was generated by a program, it is not so clear. What would happen for example if `ListPlot` was used to generate a sequence of pictures for an animation? Perhaps the animation would show the imaginary parts of numbers decreasing. But then suddenly if all the imaginary parts happened to be zero, `ListPlot` would do something completely different. The user would be justifiably confused.

Needless to say, `ListPlot` was not extended to handle complex number lists. In fact, I remembered I had thought about this possibility when I originally designed `ListPlot` and had decided at that time that it was a bad idea. But the story illustrates that getting a good design can be quite a subtle matter.

In fact, in building the whole *Mathematica* system, the part I spent longest on was the design. And if you write programs or packages in *Mathematica*, you should also be spending a significant part of your time on design. It isn't enough to have a great set of features or algorithms. You need to have a good overall design. Otherwise, nobody will be able to use what you do at anything but a superficial level. And quite possibly, you won't even be able to use it yourself if you come back to it a few months later.

I am not claiming that good design is easy to do. In fact, computer system design is certainly one of the most difficult things I have ever done. You have to understand the functionality you are trying to achieve very clearly. And, you often have to puzzle for a long time, trying to work out the simplest, most obvious way to have something work. Then you have to check that you haven't missed any strange cases, as in the `ListPlot` example above.

People often underestimate how important good design is. They think that what matters is only the set of features you have. Features are certainly nice. (How could someone responsible for a system with nearly a thousand built-in functions think otherwise?) But when it comes to really using a system, what is at least as crucial is how all those features fit together. If the features are designed properly, it is easy to create your own features from what is already there. If the features are designed poorly, all you can do is use the features already provided, one at a time.

In a sense, system design is a rather thankless task. If you get the design right, most people don't notice it. The system just "does the right thing." It is only when something goes wrong with the design that people notice.

Indeed, I am therefore rather happy that I so rarely hear comments about the design of *Mathematica*. In fact, the only consistent response I hear is from people who know *Mathematica* quite well and who try to use other systems. They never really notice the design until it isn't there any more.

So what constitutes good design? The single most important element I believe is consistency. Everything needs to work in as consistent a way as possible, so people can build up as simple a conceptual model as possible. For example, if you have functions to name, you should make the names as systematic as possible. Saving a few letters here or there may make names faster to type, but much more time is spent trying to remember the correct name than typing a few more letters.

If you write a *Mathematica* package, one of the most important kinds of consistency to maintain is consistency with the rest

Stephen Wolfram is president of Wolfram Research, Inc., and the main designer and documenter of the kernel of Mathematica. He can be reached at the Wolfram Research address, or by electronic mail at `sw@wri.com`.

of *Mathematica*. Needless to say, I believe the overall design of *Mathematica* is fairly good and worth maintaining consistency with. But more important, most people who try to use your package will already know how to use *Mathematica*. So, if your package works in a way that is consistent with *Mathematica*, people will be able to learn to use your package much more easily.

When I design *Mathematica* packages, I tend to adopt a rather extreme approach. I like to insist that every single design feature in my package have a definite precedent in *Mathematica*. *Mathematica* is a broad enough system that almost any design issue you face in a package will probably have come up somewhere in the design of *Mathematica* itself. In almost all cases, I find that by far the best thing to do is to resolve the design issue the same way *Mathematica* does.

First, some thought was probably put into how *Mathematica* resolves the issue. But more important, if you use a mechanism which people have already learned using other parts of *Mathematica*, they will not have to learn from something new to use your package. And in addition, by following the standards of other parts of *Mathematica*, you will often find that you can more easily exchange data with other functions in *Mathematica*.

Let us say you are designing a function that will use a sequence of data points. Sometimes, you just want to specify the y coordinates for these data points. Sometimes, you want to specify both x and y coordinates. What functions in *Mathematica* already have to deal with data like this? `ListPlot` is an example. It lets you specify either a list of y values, or a list of two-element lists of (x, y) pairs. You should probably make your function work just the same way. Indeed, several functions in *Mathematica* as well as `ListPlot` already work this way. Examples are `Interpolation` and `Fit`. If you use this same format for your data, you will always be able to use any of these standard *Mathematica* functions to manipulate your data — without ever having to do any kind of conversion.

As with most things, the idea of precedents must, however, be exercised with some care. If you use a precedent, you must use it fully. One of the worst things is to have a function that works almost like an existing function. People will find it very difficult to remember exactly what the differences or exceptions are. If you find a precedent that does not quite fit your needs, you should look for another precedent, rather than trying to adapt the first one you found. And, if you really cannot find any appropriate precedent at all, you should invent a completely new scheme. If, at all possible, you should make your new scheme work in a way quite different from the precedents it just missed — that way, people will be less likely to get confused. In addition, if you invent a new scheme, you should try to use it as widely as possible. People will not remember the scheme unless they see it fairly often.

I have talked of consistency within *Mathematica*. What about consistency with other knowledge people have? Obviously one should try to make things in *Mathematica* work in a way that is as close as possible to the way they work in mathematics or in some other application area. But one must recognize that *Mathematica* — and computers in general — are a new medium for expressing ideas, and sometimes one must make changes.

Of course, some existing computational usage has been much constrained by the particular limitations of FORTRAN and the like. *Mathematica* does not have these constraints, and one should not imitate them. In an example like the one above, where data could be specified either as $\{y_1, y_2, \ldots\}$ or $\{\{x_1, y_1\}, \{x_2, y_2\}, \ldots\}$, I have seen people put an extra integer parameter into their function to say which form the data will be in. No doubt, this is necessary in a FORTRAN subroutine call, but it is completely unnecessary and confusing in *Mathematica*. Or worse, I have seen people use a parameter that can take values, say one through seven, to represent different functional forms. *Mathematica* is a symbolic language: Why not just have the user give those functional forms explicitly, rather than having to remember a strange integer code for them?

Many more things can be said about good design in *Mathematica*, and perhaps they will be said in future columns in this journal. But when you have come up with a design, how can you tell if it is a good design? The best test I know is to try to document the design. You must force yourself not to gloss over important details. But, if the final description is easy to write, short, and flows well, the design is probably good. If you find it difficult to write the description, you have to cover many different cases and so on, something is probably wrong with the design.

If so, then it is time to remember what your package was originally supposed to do and to ask, "What is really the obvious way to do this?" Don't stop asking until you've figured it out. Be glad you don't have to design something with 843 functions.