++++++++++ REVIEW * (Clayton Lewis) +++++++++

I've read all of the vision statements (as of May 7, 2025) with great pleasure. Rather than commenting separately on each, I'm posting the same "review" for each submission, as most of the thoughts seem more relevant to the project of the workshop as a whole, than to the separate visions in particular.

There is clearly a great deal in common among the visions (as one would hope). At the same time, there are many divergences, at least in focus. That means there's a point to the workshop as a project: clarifying the areas of agreement and disagreement in what (since there is at least some overlap of aims) can be a useful community.

Having said that, effective communication will be crucial, and there's a long way to go on that. Tomas Petricek points to one of the challenges: the packaging of collections of ideas as demo systems. That emerged for me as a real issue in reading nearly all of the submissions.

On the one hand, embodying ideas in working systems is clearly a valuable test of their workability. On the other hand, doing so makes it hard to evaluate the ideas outside a particular context, or to compare what might be the same, or very similar, ideas that show up in different systems. Dave Thomas points out the challenges to collaboration that resulted from people splitting up to work with different protoplatforms.

Petricek calls for making demos work better for communication within the field. He says, "The key issue is establishing norms for what aspects of the system need to be explored in depth and what aspects can be ignored as unnecessary technical detail." Can one add to that? Besides this important point about focus, I'd like to know what dependencies there are among the features that a demo exercises. That is, to what extent does addressing aim A with feature F entail addressing aim A' with feature F'?

Also, I'd like to know what cognitive investments users of the demo system have to make. What do I have to know, or understand, to use the system as the designers intend? This is of course a very slippery business, partly because it is so easy for designers to overlook their own investment in understanding their own ideas. These ideas often come to seem self evident, when in fact they took months or years to mature.

There are ways to attack this problem, starting with just recognizing that it actually is a problem. One can then make an effort to articulate the budget of required knowledge or perspectives ("You have to understand that there's this thing called the DOM" or "You have to see how an AST is a more flexible representation of ..." or "You have to know that when you press a key there's an event that is generated, and you can write or edit a program called a handler..."). A slightly more structured approach to this is described in https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380240102.

A further step towards exposing the latent knowledge in the demosphere could be a bake-off process. This is a structured form of collaboration in which the proprietors of a collection of demos, that have overlapping scope, develop and share a pool of example applications. In minimal form, each participant contributes two examples, one that they feel highlights something that's especially good about their system, and one that brings out a defiency, something they wish their demo system could support, that it can't. All participants then try to address all of the examples, using their system, and describe the results. (Of course participants could share more than two examples, and more, simpler examples, rather than fewer, more complex ones could help clarify things.)

Many useful outcomes are possible. B's system may handle A's positive example with ease, showing that the issues in it can be approached successfully in different ways. B might also be able to handle A's negative example, better still. Or it may be that B's system just doesn't attempt the things that A's does, helping to define different aims within this broad field. Another kind of outcome could be that A's and B's systems can both handle an example, but A's approach can be seen to make fewer cognitive demands on users, when the two solutions are compared.

A collection of examples can act as a kind of benchmark pool for the field. Agreement could emerge that certain examples, either positive or negative, clearly capture what the field is striving to do, in some aspect of the work. Somebody presenting a new system can then easily locate its capabilities, as they compare to those of existing systems. They may also want to contribute new examples, that demonstrate something putatively valuable that others had not thought of.

Flexibility would be needed to make this process useful. Insisting that B's system hasn't done exactly what A's system has done may or may not be appropriate. Judgment should be used in deciding whether the differences are consequential. This is related to Petricek's call for judgment about what technical details are important.

I've already suggested that a bake-off would expose differences in aims, as well as differences in technical approaches, within the field. For example, some people are focussed on integrating existing software into their platform's sphere, and others are not. So the examples from different sides of this would be different. But people can be flexible in addressing only some aspects of an example, showing how there could be common ideas about editing (say) that are useful for projects of both kinds. (Subexamples to make this point could be generated and saved, to mark the overlap of concerns.)

The envisioned sharing of examples may help address an underlying problem that can limit collaboration: the tendency we all have (certainly, I have it) to prefer a world that is all of our own shaping. Aspects of academic credit assignment can make this worse (in another field, someone has said, "Psychological theories are like toothbrushes. You only want to use your own.") We like to work out our ideas in our own systems. But shared examples make it easier to coordinate work, and draw on the contributions of others, while still identifying one's "own" system.
This preference for our "own" ideas likely owes something to another issue, mentioned earlier: the dependencies among our design ideas. If you want to address problem P with method M, then you're probably going to want to address this other problem in a different way from what you might otherwise have thought. Things can seem more comfortable, or actually be more comfortable, when a whole tangle of design choices are made together. Mapping all of this into a space of shared examples can help understand underlying issues, despite the tangles.

Cheers to all, Clayton