- djrobstep.com
- posts
- programs-are-a-prison

Programs are a prison: Rethinking the fundamental building blocks of computing interfaces

We often hear that Apple's ecosystem of apps (or Microsoft's, or Google's) are "walled gardens". But what about the individual applications themselves?

In fact, individual programs are even more harmful walled gardens - a stifling barrier to true expressiveness, productivity, freedom and consistency of computing experience.

Imagining software beyond apps or webpages or programs

Think about adding up some numbers in a tabular structure. That's straightforward with most programming languages. But what if that same table is in a web page, or a mobile app, or a PDF? It's right there on the screen, it's probably encoded as a table in the markup. So the data is there. And yet, we can't query it.

Maybe if we copy-and-paste it we can query it. Or maybe we can download the page, save it to disk, write code to load the page, parse the table, and dump the data into a database.

But why should we need to do any of those things? Why can't we just query it directly?

There's a wall around each app preventing this sort of thing. A *program* is in the way, blocking our path to expressiveness and computing freedom. Much like great nations, great software should build bridges, not walls.

What would software look like, beyond standalone applications? What would software look like without walls?

A new kind of object-orientation

Notions of object-oriented programming are long-established and familiar. In the OO heyday there were high hopes that OO would allow much higher levels of

expressiveness and encapsulation, and along with it, greater reuse and recombination of code.

This absolutely hasn't eventuated - once again, applications are the problem. Photoshop's codebase and Instagram's codebase no doubt have sophisticated `Image` objects defined, but each only exists within its gated prison.

A truly expressive system would let us seamlessly apply any filter to any image, but this isn't currently possible.

Each has an entirely separate notion of what `Image` means and can do - each app essentially rebuilds an object's meaning from the ground up. If we're interested in inter-operability, re-use, and consistency, this is a terrible situation.

We need computing environments where the building blocks are inherently interoperable *objects* - to interact with directly, without the concept of applications appearing at all.

Currently the most compelling software is that which locks the most functionality within it - in an environment of objects, the most compelling software would be that which exposes the most inter-operable re-combinable functionality to the objects around it.

This is in no way a new or original idea - Alan Kay and friends in the Smalltalk scene have been playing around with such concepts for decades with things like Squeak. And similar ideas have been used quite effectively in business system implementations with naked objects, the notion that a user interface should be directly generated from the objects it represents.

Perhaps the most well-known effort to build richer, consistent meaning into the software experience is the semantic web, however its adoption has been limited.

Whatever the success of these systems, it hasn't led to more general purpose systems being developed around these concepts.

Why not? What are the obstacles and how can we overcome them?

Conway's law

Conway's law has been getting a lot of airing recently - and for good reason. It has a lot of explanatory power for the contemporary state of software.

Conway's law postulates that the structure of software reflects the social structure the software exists within - readily observed in typical corporate websites where the structure mirrors that of the corporate hierarchy rather than around the needs of the users.

But it is at the application and operating system level that Conway's law is most engrained, and most pathological.

The components of the operating system - the core OS itself, its applications, and its websites - are a manifestation of the economic entities that create them.

The boundaries have to be drawn somewhere - Mozilla creates a browser but can't create all the websites you need. Microsoft wants to let you use Windows for 3d modelling but doesn't create that software itself. Same story with the iPhone - even an enormous corporation like Apple needs external app developers to enable the iPhone's range of capabilities.

The solution is sandboxing - processes and containers, browser tabs, phone apps. The code we need for day-to-day computing sits within these sandboxes, and in doing so, imprisons that code, preventing the expressive power and consistency we've mentioned.

Often these apps will have features to integrate with other apps and the wider operating system - but not so much that they become invisible. Instagram still wants you to see its logo, consume its specific content and stay within its ecosystem. Once again, the implementation and architecture are driven by economic imperatives.

We'd like all our conversations to be in one place - but the imperatives have them split between several different protocols (often proprietary) and tens of different software clients.

The hardest technical problems to solve are political problems.

Principles for empowered computing

Better systems must be built along the following principles (for starters):

- **Widely reusable meaning:** Right now our shared "primitives" are very primitive indeed: basics like bytes and sockets. We must build much higher level shared meaning - `Images`, `Tables`, `Conversations` and beyond, building a common implementation and understanding used by everybody. This must not be limited to an optional sprinkling of additional meaning a la the semantic web - shared meaning and interoperability must be baked into the OS as fundamental.

- **Data without borders:** The "things" of computing should be able to talk directly without anything in the way. For instance, any two tables of data available to the user through the OS should be joinable or otherwise combinable as easily as if they were in the same database.

- **Inherent, ubiquitous programmability:** Currently, "doing programming" is a

> segregated activity from mainstream computing - separate software, command lines, specialist knowledge, clunky text-driven interfaces. This must end. Real expressiveness demands that every entity in the interface is inherently programmable - a table of data shouldn't just be a rendered picture of a table of data - it should *be a table*. Programming shouldn't be *separate* at all.

Building something different

There's no doubt these concepts are very broad and open-ended. Beyond the political problem of challenging status quo computing and pushing for an alternative, there are complex design and technical challenges in the specifics.

How do you build ubiquitous programmability into interfaces without adding clutter or reducing usability? How do you remove barriers between programs and more tightly integrate data without compromising security?

Conway's law tells us that to build something different, we need to build upon different social and political structures.

Open-ended research

Short-term commercial realities mean packaging things up in sellable units - escaping the application prison requires the opposite approach. Developing a fresh approach is going to require isolation from the short-term imperatives of the market.

Academia used to offer this. But the relentless march of neoliberalism has seen the open-ended possibilities of research constrained by "business logic", funding processes that demand narrow parameters of investigation, and a creativity-crushing increase in busywork. Peter Higgs of Higgs boson fame famously commented that he wouldn't be productive enough in today's academic environment.

The other bastion of open-ended research was government - the internet itself came from US military research. Here too funding for long term open-ended research has dried up, and even the military money has been more tightly constrained to projects with more obvious, shorter term military application - our amoral powerful would rather spend a trillion building new fighter jet systems than new software systems.

Rare-but-notable efforts like Xerox PARC suffered similar fates, able to fend off the bean-counters for a while but not indefinitely.

Rebooting

The realization that the software experience is still built on artifacts of computing from the 80s like text-based command lines is a lot less surprising considered within the context of this ongoing decline.

Climate change has shown us that mere awareness of the situation we are in isn't enough. Actual liberation from disaster requires a bold change of direction and a acknowledgement of shared, public goals beyond the financial.

Liberation from the constraints of today's limiting computing concepts will require similar boldness - large scale open-ended research, long term timeframes, but most fundamentally, a willingness to change path.

Next steps

If you have thoughts on these ideas or how to make them happen, I'd love to hear from you. Tweet at me.

If you liked this you might also be interested in my talk: Instant feedback, instant debugging Python coding

djrobstep.com | twitter | github