On Visual Programming

Text vs. A More Visual Representation

Firstly, note that indentation IS visual programming. Textual programming, as we practice it today, contains a huge dollop of visualization.

Experiment: remove all indentation from a piece of code and look at it. Some kind of information is gone. I don't think that we've characterized *what* that information is, but, it's obvious that it is useful information of some kind, even though we can’t put our fingers on what it exactly is.

We are working with 4 dimensions (x/y/z/t). Our notation, and, our analytic techniques, and, our brains, can handle only 2 dimensions at a time. That implies that we *must* use more than one notation to cover all of the dimensions. For example, famous physicist Feynman needed to stop using functional notation and resorted to using Feynman diagrams to analyze a certain aspect of physics. We keep seeing new programming languages spring up. I think that this is a manifestation of the fact that 2D text is not expressive enough to handle a union of all 4 dimensions in one big gulp.

Does that mean that we should give up on text?

No.

But, it does mean that we must use *many* notations, some text, some other, to address the various issues that crop up in programming.

Right now, we are walking off of a cliff in programming from single-unit to multiple-unit machines, i.e. distributed programming - robotics, internet, blockchain, Arduinos, even NPCs, etc., etc. We're discovering that our current function-based thinking is tying us down when we want to express these kinds of problems.

Can express is different from *convenient* to express. Language affects thought. If it really doesn't matter, then the obvious conclusion is that we should stop using Haskell and go back to writing all programs in raw assembler.

Our current notations[1] all hard-wire synchronization into the notations at a very low level. Sync restricts variance along one dimension, but, leaves out a lot of possibly useful analyses[2].

"{...}" is ASCII for "*box".* It should mean *closed*-box, but, it doesn't. "{...}" ain't enforced as a closed-box and programmers have, ingeniously, figured out work-arounds. Programmers run/ran into gotchas like the *global variable* problem. I doubt that programmers would create global variables if they would draw their programs on whiteboards using rectangles. The global-ness of variables is obvious on a whiteboard, but, less obvious when written out as big lumps of text.

What other gotchas are lurking in the shadows of text-only notations? Just ask the rhetorical question "What is currently considered to be difficult?". If something is considered to be difficult, e.g. so-called "essential complexity", then it is probably not really difficult at all, except that we're using the wrong notation to deal with it.

Off of the top of my head, I'd say that concurrency is being mis-handled by text-only notation and could use fresh ideas on how to jailbreak the concepts into some new non-textual, non-function-based notations.

[1] Python, Javascript, Haskell, Rust, etc., etc.

[2] Note that Christopher Alexander wasn't thinking that his architectural units were strongly-coupled, i.e. synchronized, but, programmers have been trying to express his ideas using only strongly-coupled programming languages. Square peg, round hole.

Appendix - See Also

References

https://guitarvydas.github.io/2024/01/06/References.html

Blogs

https://guitarvydas.github.io/

https://publish.obsidian.md/programmingsimplicity (see blogs that begin with a date 202x-xx-xx-)

Videos

https://www.youtube.com/@programmingsimplicity2980

Books

leanpub'ed (disclaimer: leanpub encourages publishing books before they are finalized - these books are WIPs)

https://leanpub.com/u/paul-tarvydas

Discord

https://discord.gg/Jjx62ypR

all welcome, I invite more discussion of these topics, esp. regarding Drawware and 0D

Twitter

@paul_tarvydas

Mastodon

(tbd, advice needed re. most appropriate server(s))