Review of: "Channeling the Flow — A Metaphor for Computer Programs"

Valeriy Mygal[1]

1 National Aerospace University Kharkov Aviation Institute

Potential competing interests: No potential competing interests to declare.

Dear Attila Egri-Nagy

The article provides interesting analogies, comparisons, and visions of problems, which truly enrich the experience of computer programming and deepen its understanding. Therefore, the main goal of the work has been achieved. It is important to note that the author develops three ideas, the complementarity of which has heuristic and cognitive value.

The first idea is the relationship between dynamics and statics. Shifting the perspective from the creation of a computing process to the formation of an existing one, the author views software as a static entity. This philosophical approach allows us to focus on the dynamic aspect and draw attention to the fact that by running code on program launch, we limit capabilities and reduce the degree of freedom. The author imagines limitation as an attempt to control something that is already in motion. There are two interrelated aspects of calculations here. On the one hand, the dynamics of the physical system matter, and on the other hand, the final result (software) determines the static essence. Therefore, a program is a set of restrictions that act as barriers to a previously existing aimless process, and programming is the ability to correctly place obstacles or the “art” of removing unnecessary things.

In my opinion, the choice of the main example is good - artificial waterways. This is a visual and physical example of how we think metaphorically about programming.

Conceptual metaphors organically organize our everyday human experience, and the author uses these cognitive tools explicitly as tools for transferring knowledge between different domains and searches for a functor (Conservation of Structure) between two categories by studying the extent to which the internal structure and dynamics of one domain correspond to another domain.

The second idea is to expand the concept of universality through everyday experience, because there is mathematical evidence to establish the equivalence of different models of calculations and this universality. If a computer can do something computable, then there is no directionality in computing. After all, program execution is also a form of movement: a running processor takes a kind of walk in the space of dynamic states of the computer, which is determined by the totality of all possible memory configurations and internal registers of the processor. Technically, the dynamic state space would be a graph with cycles, but we can model it using a tree for input-output computations that ensure steady progress toward achieving results. This tree structure, with its extremely high branching ratio, can represent all possible processor state transition dynamics starting from a given reset state.

The second idea is to expand the concept of universality through everyday experience, because there is mathematical evidence to establish the equivalence of different models of calculations and this universality. If a computer can do something computable, then there is no directionality in computing. After all, program execution is also a form of movement: a running processor takes a kind of walk in the space of dynamic states of the computer, which is determined by the totality of all possible memory configurations and internal registers of the processor. Technically, the dynamic state space would be a graph with cycles, but we can model it using a tree for input-output computations that ensure steady progress toward achieving results. This tree structure, with its extremely high branching ratio, can represent all possible processor state transition dynamics starting from a given reset state.

The third idea is the complementarity of inversion and cognitive metaphor. The author writes, “We used channels, engineering artifacts, as a metaphor for programming. However, the cognitive metaphor can be reversed. Civil engineering precedes software engineering, and nothing prevents us from using the latter to understand the former, reversing the chronological order. Perhaps someone trained as a programmer used their knowledge of computers to project it onto the physical world to understand engineering. Thus, we have the metaphor of engineering as programming the physical world."

In programming, optimization is aimed at removing unnecessary calculations or trying to reduce the amount of data transferred.

When the abstract computational meets the physical world, the general principle of efficiency is important.

The author points out that “Given the similarities between design in the physical world and software development in the abstract, there is an interesting middle ground between virtual worlds. Sandbox-style video games like Minecraft define a virtual world with its own “physical” laws. Players use these laws by placing blocks in a discrete world."

Overall, the article is quite well structured and contains three new ideas. The conclusions are clear and consistent with the main text. The manuscript is written clearly and may be of interest to researchers in this field, and I recommend publishing it in its current form. The article is a good approach for teaching and learning basic programming.

When developing new ideas, I recommend that the author use heuristic and cognitive metamodeling to improve understanding and reinforce key concepts. I note that cognitive visualization effectively complements the text and is a visual anchor for abstract ideas and improves the overall learning process (see https://doi.org/10.26855/er.2022.04.001.), and also promotes creative activity (https://doi.org/10.32388/GIJ3RI)

Sincerely, V. Mygal.