

Designing Tools *for* DESIGN SOFTWARE

A manifesto for programmers written by a designer

Designing an intelligent and intuitive design software that closely mirrors the intentions, thought process and workflows of creative professionals is notoriously difficult. Often, this is a war between design professionals and programmers. Few designers are programmers. Fewer programmers are designers. They don't think in similar terms; share similar concepts; or evaluate themselves against similar benchmarks.

While programmers have some idea of the end user's needs and goals, their answers to these needs and goals often focus on pure, brute performance. If a software can achieve the designer's goals without the overall performance suffering in any obvious way (remember that fan whirring sound?), a programmer would consider this a good day. But the story a designer would tell is quite different.

The designer would tell the programmer that mere performance counts for nothing if the actual workflow is a mess. A good design software is not merely performant but also deeply intuitive *to the designer*. It mirrors the intentions, the mental steps designers have to take to get to their desired result.

Of course, designers are shaped in their intuitions as much by design software as design software are shaped by designers. But every designer worth his salt would argue that this mutual shaping is heavily biased in the favour of design software. Entire generations of digital-first designers have cultivated poor intuitions by working with substandard software. Software that are performant, but ultimately, pardon the language, fucking dumb.

This manifesto is written by a designer with intention of making common cause with programmers; of finding a common language that we can both understand. It is not a technical language. It is plain English. Plain-speaking is so direly missing in both our worlds. So let's try.

Here's how the manifesto is structured: First, it will set out the terms and the principles. Then, it will demonstrate through a simple visual matrix, how these terms and principles may be applied for our mutual benefit.

The Terms

Or words we can share

1. **Unify:** Merge actions into a single tool to reduce the need for tool switching.
2. **Modify:** Allow actions to be done with a different tool by calling it temporarily through modifier keys.
3. **Oppose:** Allow a tool to perform the opposite action of its normal action temporarily through modifier keys.
4. **Group:** Group tools into a single tool setting or dialogue box to reduce the need for navigation.
5. **Stay:** Make dialogue boxes and tools settings to stay on screen (non-modal) until the user chooses to close, whilst allowing interaction with the rest of the software.
6. **Direct:** Allow a tool to be used directly and dynamically on the screen, without the need for extra

tool settings and dialogue boxes for at least the first set of actions.

7. **Even:** Make different tools to act in the same way.
8. **Extend:** Make the functions of one tool to be available to a different tool.
9. **Expand:** Introduce new modes of operations to a tool.

Those are, in my opinion, the only 9 terms we need to have a common language. They're simple enough for designers and programmers to understand and wide enough to accommodate everything they need to work together.





































So what are the principles?

Principles

or guardrails to work together

1. If two or more tools do logically or sequentially related actions, unify such tools.
2. If usability would suffer by unifying tools that perform logically or sequentially related actions, then allow the tools to be temporarily called up with modifier keys.
3. If two or more tools are logically or sequentially related, but cannot be unified, then group such tools in to fly into a single tool setting or dialogue boxes.
4. If two tools perform logically related but opposite actions, then allow the tools to be called up with modifier keys.
5. If a dialogue box or tool setting will be accessed repeatedly as a part of a series of actions, then allow such tools to stay on screen until the user chooses to close it and allow interaction with the rest of the software.
6. If an action requires immediate visual feedback, make the tools for doing these actions direct and interactive on screen.
7. If there are two or more related tools, which are neither united, nor modified, nor opposed, make them act in the same way.
8. If two or more tools are similar but not logically or sequentially related, then make the functions of one tool available to others.
9. If a tool can be made more useful either through additional modes or compatibilities, add such modes and compatibilities.

And those are all the 9 principles we'll ever need, in my opinion. Now, let's see how you can turn these principles into useful actions for programmers.

Relationship	Logically related	Sequentially performed	Iteratively performed	Mutually complementary	Similar in action	Current Solution	Tool simplification
Actions							
Create vector shapes						Curve Creation	
Create straight lines						Create Line	Modify, unify or expand with curve creation and vector drawing tools
Draw vector shapes						Vector Drawing	Expand with curve creation tool
Move vector points						Curve creation & Edit vector points	Unify with curve creation tool
Move vector paths						Edit vector points & Transform vector points	Unify with curve creation
Add vector points						Edit vector points & insert vector points	Group and oppose with curve creation and create line tools
Delete vector points						Edit vector points Delete vector points	Group and oppose with curve creation and create line tools
Move vector point handles						Curve creation & Edit vector points	Unify with curve creation
Transform vector points						Transform vector points	Modify with edit vector points tools
Covert to smooth & sharp						Covert vector points	Unify or modify with curve creation, create line and vector drawing tools
Align vector points						Automatic	Make even: Allow object alignment for vector points too
Join vector paths						Connect vector paths	Group and oppose with cut vector path tools
Cut vector paths						Cut vector path	Group and oppose with join vector path tools
Optimize Vector layer						Optimize Vector layer	Stay and direct: Make this available as a slider with the curve creation and vector drawing tool settings.
Create vector fill						Flood tool	Make even: Allow flood fill to be available within vector tool bar