<https://martinfowler.com/bliki/InternalReprogrammability.html>

# [InternalReprogrammability](https://martinfowler.com/bliki/InternalReprogrammability.html)

*[](https://martinfowler.com/)*

[*Martin Fowler*](https://martinfowler.com/)

*10 January 2013*

I was programming away and wanted to add an empty line above where I was currently typing. The editor I was using doesn't have this feature built-in, and I'd finally had this desire enough that I really wanted it. I did a quick google search, found a few lines of code, pasted them into my startup file, executed them, and lo I could now create empty lines above with a single keystroke. It took just a couple of minutes, I didn't have to install any plugins, or restart the editor - this is normal everyday business for an emacs user.

Emacs is an elderly piece of software, [dating back to the mid 70's](http://www.jwz.org/doc/emacs-timeline.html). Its philosophy of allowing people to easily extend it by modifying the live environment is something shared with a few other elderly-but-groundbreaking systems, such as lisp machines and Smalltalk.

That philosophy seems rarer now. Certainly there are plenty of extensible systems, you can install plugins for browsers like Firefox and editing suites like Eclipse. The whole free/open source movement is about giving you access to the code that runs your machines so you can (in theory) tweak it to your heart's content. But there's a palpable difference between extensions in most of these environments and the kind of reprogramming you do in emacs or Smalltalk. Something about how it's easy to quickly do small modifications, such as the new command I added above. It's also about doing it without leaving the environment - I don't fire up some separate toolchain to add an emacs function, I work within emacs itself.

This is also different to tools that add some kind of "macro capability". Adding a new elisp function is exactly how emacs is programmed itself - there is no difference between how you program little extensions and the core programming of the software. This unity allows you to reach deep into the editor's guts. It also means that your modifications aren't relegated to some "scripts" menu - they are indistinguishable from any other part of the tool.

This capability is also a philosophy about how you relate to your tools. For many people the software you use is a relatively fixed product. Even plugins add a relatively limited menu of options. Internally reprogrammable tools allow you to add or change any part of your software, allowing you to craft your tools to exactly fit your metaphorical hand.

This thinking even applies to programming languages themselves. Both Lisp and Smalltalk are minimal languages that make it easy to extend the language in such a way that extensions look identical to the core. Neither language has any special syntax for such basic language features as conditional logic. This flexibility allowed Smalltalk to add exception handling without any language changes.

One of the biggest issues with internal reprogrammability is the resulting fragmentation of instances of the software. As I modify my emacs instance with lots of personal functions, I'm creating my own custom version of emacs that's tightly coupled to the emacs configuration on my machine. Inevitably this raises questions about dealing with upgrades to the core application and on how easy it is to share my functions with others.

Systems with plug-in architectures and macro languages handle this by reducing the surface area of customization, but as [Nic Ferrier](http://twitter.com/nicferrier) put it well: "A reprogrammable system is incredibly powerful. Abusing the power is always possible and it's a point of principle in a reprogrammable system that people *must* be able to abuse it."

The emacs community is, of course, a good example of how this has progressed in practice. Emacs has stabilized enough that, despite regular updates, most people are able to upgrade without serious headaches. Emacs has used package management systems to help distribute sharable changes - there's been much progress in thinking about how to share code since the original Emacs and Smalltalk days. The rise of distributed version control tools adds more ideas for managing shared code.

Despite the sense that internal reprogrammability is a mostly-forgotten philosophy, there are some interesting flickers of life. At [gotoAarhus2011](https://martinfowler.com/bliki/gotoAarhus2011.html)I was struck by how well [Moose](http://moosetechnology.org/) (a tool for assessing codebase health) used reprogrammability to help you gradually build up health-check analysis. Emacs is still widely used and being pushed into new frontiers. A group of emacs developers have built a [node-like web-server in emacs](http://nic.ferrier.me.uk/blog/2012_08/elnode-nears-1-point-0). Ward Cunningham's recent thinking on developing the [smallest federated wiki](https://github.com/WardCunningham/Smallest-Federated-Wiki) includes a strong element of reprogrammability through the ability to [create your own javascript functions](http://vimeo.com/33131381) to manipulate data on your pages.

If internal reprogrammability is rare for tools aimed at programmers, it's even rarer for tools aimed at non-programmers. I've often wondered if that ought to change. What would come from making more tools exhibit this quality? Would this encourage more people to learn about programming, the better to control the environment that they spend so much time in? This was certainly part of [Alan Kay's vision of the dynabook](http://www.mprove.de/diplom/gui/kay72.html). He saw children not as passive consumers of media, but actively programming their environment.

Programming is not easy, and I'm not one to underplay the challenges programmers face every day. But that doesn't mean that internal reprogrammability should be relegated to 1970's vision of the future. A large part of why modern dynabooks lack the internal reprogrammability of Kay's vision is because it hasn't been made a high-enough priority. Perhaps that's something we should think about more.

## Acknowledgements

I really appreciated the conversation on our internal mailing list between [Nic Ferrier](http://twitter.com/nicferrier), [Pat Kua](http://www.thekua.com/atwork/) and Kief Morris on the issues around fragmentation.