ByteNoise

Modular

Modular systems are ones which comprise many simple components rather than one daunting one. These components can be combined in increasingly complex ways as your familiarity of the system grows, allowing you to achieve more and more with the same system. For examples of good modular systems, look at the Moog modular synthesizers or the Unix operating system.

The Moog modular synthesizer

Arguably the ancestor to pretty much every modern synthesizer, the Moog modulars were vast machines that took up whole walls and involved connecting all the modules together using patch cables (this is why each different sound a modern synthesizer makes is stored as a "patch": you are really storing the information of which outputs are patched into which inputs, as well as the settings for all the virtual dials). They were very versatile instruments that implemented subtractive synthesis in a reasonably simple way by breaking it down into elementary parts.

You start off with an oscillator, which makes a musical buzzing noise that the NES would have been proud of. You can then feed its output into the input of a filter, which cuts off the higher frequencies to make it sound less harsh. You can then, in turn, feed the filter's output into an attenuator which changes the

volume of the noise, and finally feed that into a speaker. The filter and attenuator also have VC inputs, which let you modify their behaviour over time. Hooking up a envelope generator, such as an ADSR envelope, to the VC input of these will make both the overall volume and the brightness of the sound rise then fall back down again over time. Using just very basic components - an oscillator, a filter, an attenuator and an envelope generator - in a certain way makes them form a system much more powerful than the sum of its parts.

The Unix pipe

Over thirty years old and still going strong, Unix implements many features that make a lot of sense, not least of which is the Unix pipe. It works in the exact same way as patch cables work on the Moog modulars, allowing you to transfer the output of one simple program into the input of another.

Say, for argument's sake, that you want to check the spelling of all the words in your main web page, `index.html`. You can do this using the command `spell -H index.html`. That's all well and good, but as `spell` is a simple program, it lists all the spelling mistakes in the order they occur. This can sometimes give you an overwhelming amount of information, often showing that you made the same mistake two or three times. What if you only want to see one instance of each mistake? You simply pipe the output of `spell` into another simple program, `sort`. The command `sort -u` takes all the lines you give it, alphabetises them, then removes any duplicate lines. So the command `spell -H index.html | sort -u` will list all your spelling mistakes, but only list each mistake once. If the mistakes are *still* taking up more than a whole screen, you can even pipe `sort`'s output into `less`, letting you view one screen's worth of

information at a time. This combining of simple tools can make complex procedures relatively easy to automate, letting the computer do all the repetitive work for you (which is surely what they were designed for in the first place).