October 26, 2018 — 3 min read

Small Focused Modules

Make small focused modules for reusability and to make it possible to build larger more advanced things that are easier to reason about.

This was originally an answer on my AMA (Ask Me Anything) about why I make small Node.js modules. My answer applies to Node.js and might not be applicable to your platform. The reason this works so well on Node.js is that you can have nested dependencies, so they never conflict.

People get way too easily caught up in the LOC (Lines Of Code). LOC is pretty much irrelevant. It doesn't matter if the module is one line or hundreds. It's all about containing complexity. Think of node modules as Lego blocks. You don't necessarily care about the details of how it's made. All you need to know is how to use the Lego blocks to build your Lego castle. By making small focused modules you can easily build large complex systems without having to know every single detail of how everything works. Our short-term memory is finite. In addition, by having these modules as reusable packages, other people can reuse them, and when a module is improved or a bug is fixed, every consumer benefits.

Imagine if PC manufacturers all made their own CPUs. Most would do it badly. The computer would be more expensive and we would have slower innovation. Instead most use Intel, ARM, etc.

This would not be possible if it weren't for how npm works. The beauty of being able to use nested dependencies means I don't have to care what dependencies a dependency I use have. That's powerful.

Some years ago. Before Node.js and npm. I had a large database of code snippets I used to copy-paste into projects when I needed it. They were small utilities that sometimes came in handy. npm is now my snippet database. Why copy-paste when you can import it and with the benefit of having a clear intent. Fixing a bug in a snippet means updating one module instead of manually fixing all the instances where the snippet is used.

For example, chalk is one of the most popular modules on npm. What you might not realize is that it's actually a collection of modules. It depends on a module for detecting if the terminal supports color, for getting the ansi escape codes, etc. All of this could have been just embedded in the main module, and it probably is in many cases. But that would mean anyone else wanting to create an alternative terminal string styling module would have to reinvent the wheel on everything. By having these supporting modules, people can easily benefit from our work in Chalk and maybe even help improve Chalk indirectly by improving one of the dependencies.

Yet another example. I have this module user-home which gets the user's home directory. You might think it would be simpler to just do `` `process.platform === 'win32' ? process.env.USERPROFILE : process.env.HOME` ``. And most do this. But first, why require everyone to know how to get the home directory? Why not use a "Lego block"? What you also might not realize is that this check is incomplete. On Windows, you should also check `` `process.env.HOMEDRIVE + process.env.HOMEPATH` `` and you might also want to do additional checks. Lego blocks. (Node.js actually has an API for this now, exactly for the reasons stated above)

Do you make your own shoes? No, you buy them in a store. Most don't care how the shoe is made. Just how good it fits.

I want programming to be easier. Making it easier to build durable systems. And the way forward, in my point of view, is definitely not reinventing everything and everyone making the same stupid mistakes over and over.