The node.js philosophies

small core →

Node.js runtime and built-in modules
which contains very small set of features - **leaving the rest to the so-called userland (or userspace)**

Small modules →

designing small modules (and packages)
in terms of raw code size and in terms of scope

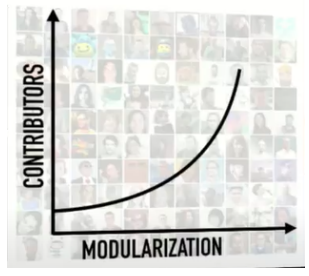
for complex projects

- the code grows in complexity quickly
- so it is important to divide the code based on their responsibility into separate chunks (modules).
- this (division/modularization) can also help in introducing new contributors and restricting them to easier parts of the code first.

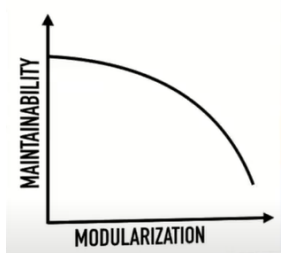
smaller modules/scopes -
- easier to document only that scope.
- easier to test only that scope
- easier to understand and hold in the human brain

Unix philosophy

- **small is beautiful**
- **make each program do one thing well**



modularization helps atracting and retaining contributors



reduces maintainabilty as managing those many dependencies is hard

Release and updates →

As the modules grow
maintaining the modules become difficult as

- update to package/module needs a chain of updates

(also the packages can be used in the **dependency-chain** both vertically and horizontally)

updating the whole chain mannualy

- is a repetitive and
- cumbersome task and
- also error-prone and
- costly

SOLUTION

- **semantic and automatic releases**
  semantic versioning tells other packages what kind of changes the package contains

to solve the dependency hell problem by making sure that two (or more) packages depending on different versions of the same package will use their own installations

NPM →

- **./npm-init.js** enforces liscense and best practices
- **npm** automaticaly **syncs** with **git**

OFFLINE

- npm global cache at **.npm/**
- npm offline **-> npm install --cache-min 999999**
- **npm pack**
  npm install recently-packed.tar.gz
  but npm pack does not add dependencies todo so add your dependecies into **bundledDependencies**

LOCK

- lock down deps - **npm shrinkwrap** deploy exact version of your primary and child dependecies **- helpful for deployoing production**
- update shrinkwrap **-> --save** & **--save-dev**

VERSIONING

- **npm version <type> -** version bumps
  <type> -> **major, minor, patch**

SCRIPTS

this gives access to diffent npm lifecycle events

- **npm run <anything>**
  this helps in managing npm lifecycle events
- **config** and **package.json** are available to the scripts

SCOPE

scope(name space) your packages

- **npm i @scope/package**
  scope packages are private by default, so, when publish
- **npm publish --access=public**

PRIVATE

use private packages

- **npm init --scope=<username>**

ORGANIZATION

grant and revoke access to npm using scopes

- **npm team**

ON-SITE

Run your own on-premise npm registry.
npm CLI client can point to a number of registry.

- **npm login --registry=http://my-npm-registery.com**