- Emacs Lisp code is easy to document. A function's definition can include a *docstring* (for "documentation string"), text explaining the function's purpose and use. Almost every function provided with Emacs has a docstring, meaning that help on a given facility is never more than a command away. And when Emacs displays a function's docstring, it includes a hyperlink to the function's source code, making Lisp code easy to browse as well. (Naturally, docstrings aren't adequate for large Lisp packages, so those usually include a more traditionally structured manual as well.)
- Emacs Lisp has no module system. Instead, established naming conventions help unrelated packages avoid interfering with each other when loaded into the same Emacs session, so users can share packages of Emacs Lisp code with each other without the coordination or approval of Emacs's developers. It also means that every function in a package is visible to other packages. If a function is valuable enough, many other packages may come to use it and depend on the details of its behavior.

  Oddly, this is not nearly as much of a problem as one might expect. One might hypothesize that Emacs Lisp packages are mostly independent, but of the roughly 1,100 Lisp files included in the standard Emacs distribution, 500 of them have functions used by some other package. My guess is that the maintainers of the packages that are distributed along with Emacs only take care to remain compatible with other such packages, and that those developers form a sufficiently tightly knit group that it's practical to negotiate incompatible changes as you make them. Packages not included with Emacs probably just break—creating an incentive for developers to get their packages included and join the cabal.

Creeping Featurism

Emacs's creeping featurism is a direct consequence of its architecture. Here's the life cycle of a typical feature:

1. When you first notice a feature that would be nice to have, it's easy to try implementing it; Emacs's lack of bureaucracy makes the barrier to entry extremely low. Emacs provides a pleasant, interactive environment for Lisp development. The simple buffer Model and automatic display update let you focus on the task at hand.
2. Once you have a command definition that works, you can put it in your *.emacs* file to make it available permanently. If you use it frequently, you can include code there to bind it to a key.
3. Eventually, what began life as a single command may grow into a suite of commands that work together, at which point you can gather them up in a package to share with your friends.
4. Finally, as the most popular packages come to be included in the stock Emacs distribution, its standard feature set expands.