Creeping Featurism and Maintainability

Obviously, the more code you have, the more effort it takes to maintain it. When a developer's Lisp package is selected for inclusion in the standard Emacs distribution, the lead maintainers invite that package's author to continue maintaining it, so as the number of packages expands, the number of maintainers expands to match. If someone relinquishes responsibility for a package (perhaps because she's too busy or no longer interested), then the lead maintainers must either find a new volunteer or remove the package.

The key to this solution is that Emacs is a collection of packages, not a unified whole. In a sense, the dynamics of Emacs maintenance more closely resemble those of a platform (such as an operating system) than a single application. Instead of a single design team choosing priorities and allocating effort, there is a community of self-directed developers, each pursuing his own ends, and then a process of selection and consolidation that brings their efforts into a single release. In the end, no single person bears the weight of maintaining the entire system.

In this process, the Lisp language acts as an important abstraction boundary. Like most popular interpreted languages, Emacs Lisp largely insulates code from the details of the Lisp interpreter and the underlying processor architecture. Likewise, the editing primitives available to Lisp conceal the implementations of buffers, text properties, and other editing objects; the characteristics visible to Lisp are, for the most part, restricted to those the developers are willing to commit to supporting in the long term. This gives Emacs's core of C code a good deal of freedom to improve and expand, without the risk of breaking compatibility with the existing body of Lisp packages. For example, Emacs buffers have acquired support for text properties, overlays, and multiple character sets while remaining largely compatible with code written before those features existed.

Two Other Architectures

Many applications allow user extensions. Extension interfaces appear in everything from collaborative software development website systems (such as Trac plugins) to word processing software (Open Office's Universal Network Objects) to version control software (Mercurial's extensions). Here I compare Emacs with two architectures that support user extensions.

Eclipse

Although most people know Eclipse as a popular open source integrated development environment for Java and C++, Eclipse proper includes almost no functionality. Rather, it is a framework for plug-ins, allowing components that support specific aspects of development—writing Java code, debugging a running program, or using version control software—to communicate easily with each other. Eclipse's architecture allows programmers with a solid solution to one part of the problem to join their work with others' in order to produce a unified and full-featured development environment.