What Unix Gets Wrong

For a design that dates from 1969, it is remarkably difficult to identify design choices in Unix that are unequivocally wrong. There are several popular candidates, but each is still a subject of spirited debate not merely among Unix fans but across the wider community of people who think about and design operating systems.

Unix files have no structure above byte level. File deletion is irrevocable. The Unix security model is arguably too primitive. Job control is botched. There are too many different kinds of names for things. Having a file system at all may have been the wrong choice. We will discuss these technical issues in Chapter 20.

But perhaps the most enduring objections to Unix are consequences of a feature of its philosophy first made explicit by the designers of the X windowing system. X strives to provide "mechanism, not policy", supporting an extremely general set of graphics operations and deferring decisions about toolkits and interface look-and-feel (the policy) up to application level. Unix's other system-level services display similar tendencies; final choices about behavior are pushed as far toward the user as possible. Unix users can choose among multiple shells. Unix programs normally provide many behavior options and sport elaborate preference facilities.

This tendency reflects Unix's heritage as an operating system designed primarily for technical users, and a consequent belief that users know better than operating-system designers what their own needs are.

> This tenet was firmly established at Bell Labs by Dick Hamming[5] who insisted in the 1950s when computers were rare and expensive, that open-shop computing, where customers wrote their own programs, was imperative, because "it is better to solve the right problem the wrong way than the wrong problem the right way".
>
> -- Doug McIlroy

But the cost of the mechanism-not-policy approach is that when the user *can* set policy, the user *must* set policy. Nontechnical end-users frequently find Unix's profusion of options and interface styles overwhelming and retreat to systems that at least pretend to offer them simplicity.

In the short term, Unix's laissez-faire approach may lose it a good many nontechnical users. In the long term, however, it may turn out that this 'mistake' confers a critical advantage — because policy tends to have a short lifetime, mechanism a long one. Today's fashion in interface look-and-feel too often becomes tomorrow's evolutionary dead end (as people using obsolete X toolkits will tell you with some feeling!). So the flip side of the flip side is that the "mechanism, not policy" philosophy may enable Unix to renew its relevance long after competitors more tied to one set of policy or interface choices have faded from view.[6]

---

[5] Yes, the Hamming of 'Hamming distance' and 'Hamming code'.

[6] Jim Gettys, one of the architects of X (and a contributor to this book), has meditated in depth on how X's

laissez-faire style might be productively carried forward in *The Two-Edged Sword* [Gettys]. This essay is well worth reading, both for its specific proposals and for its expression of the Unix mindset.
