Third-party support

An external control protocol had been developed, enabling other devices to control the Metropolis remotely. Since it was a thin veneer over the guts of the software, it reflected the Metropolis's architecture, which means that it was baroque, hard to understand, prone to fail randomly, and impossible to use. Third-party engineers' lives were also made miserable by the poor structure of the Metropolis.

Intra-company politics

The development problems led to friction between different "tribes" in the company. The development team had strained relations with the marketing and sales guys, and the manufacturing department was permanently stressed every time a release loomed on the horizon. The managers despaired.

NOTE

The consequence of a bad architecture is not constrained within the code. It spills outside to affect people, teams, processes, and timescales.

Clear requirements

Software archaeology highlighted an important reason that the Messy Metropolis turned out so messy: at the very beginning of the project *the team did not know what it was building*.

The parent startup company had an idea of which market it wanted to capture, but didn't know which kind of product to capture it with. So they hedged their bets and asked for a software platform that could do *many* things. *Oh, and we wanted it yesterday.* So the programmers rushed to create a hopelessly general infrastructure that could potentially do many things (badly), rather than craft an architecture that supported one thing well and could be extended to do more in the future.

NOTE

It's important to know what you're designing before you start designing it. If you don't know what it is and what it's supposed to do, *don't design it yet*. Only design what you know you need.

At the earliest stages of Metropolis planning there were far too many architects. With woolly requirements, they all took a disjoint piece of the puzzle and tried to work on it individually. They didn't keep the entire project in sight as they worked, so when they tried to put the puzzle pieces back together, they simply didn't fit. Without time to work on the architecture further, the parts of the software design were left overlapping slightly, and thus began the Metropolis town planning disaster.