One of the most subtle yet serious Metropolis problems was duplication. Without a clear design and a clear place for functionality to live, wheels had been reinvented across the entire codebase. Simple things like common algorithms and data structures were repeated across many modules, each implementation with its own set of obscure bugs and quirky behavioral traits. Larger-scale concerns such as external communication and data caching were also implemented multiple times.

More software archaeology showed why: the Metropolis started out as a series of separate prototypes that got tacked together when they should have been thrown away. The Metropolis was actually an accidental conurbation. When stitched together, the code components had never really fit together properly. Over time, the careless stitches began to tear, so the components pulled against one another and caused friction in the codebase, rather than working in harmony.

NOTE

A lax and fuzzy architecture leads to individual code components that are badly written and don't fit well together. It also leads to duplication of code and effort.

Problems outside the code

The problems within the Metropolis spilled out from the codebase to cause havoc elsewhere in the company. There were problems in the development team, but the architectural rot also affected the people supporting and using the product.

The development team

New recruits coming into the project (like myself) were stunned by the complexity and were unable to come to grips with what was going on. This partially explains why very few new recruits stayed at the company for any length of time—staff turnover was very high.

Those who remained had to work very hard, and stress levels on the project were high. Planning new features instilled a dread fear.

Slow development cycle

Since maintaining the Metropolis was a frightful task, even simple changes or "small" bug fixes took an unpredictable length of time. Managing the software development cycle was difficult, timescales were hard to plan, and the release cycle was cumbersome and slow. Customers were left waiting for important features, and management got increasingly frustrated at the development team's inability to meet business requirements.

Support engineers

The product support engineers had an awful time trying to support a flaky product while working out the intricate behavioral differences between relatively minor software releases.