In this chapter, I tell the story of two such software cities. It's a true story and, like all good stories, this one has a moral at the end. They say *experience is a great teacher*, but other people's experience is even better—if you can learn from these projects' mistakes and successes, you might save yourself (and your software) a lot of pain.

The two systems in this chapter are particularly interesting because they turned out very differently, despite being superficially very similar:

- They were of similar size (around 500,000 lines of code).
- They were both "embedded" consumer audio appliances.
- Each software ecosystem was mature and had gone through many product releases.
- Both solutions were Linux-based.
- The code was written in C++.
- They were both developed by "experienced" programmers (who, in some cases, *should* have known better).
- The programmers themselves were the architects.

In this story, names have been changed to protect the innocent (and the guilty).

The Messy Metropolis

Build up, build up, prepare the road! Remove the obstacles out of the way of my people.

—Isaiah 57:14

The first software system we'll look at is known as the Messy Metropolis. It's one I look back on fondly—not because it was good or because it was enjoyable to work with, but because it taught me a valuable lesson about software development when I first came across it.

My first contact with the Messy Metropolis was when I joined the company that created it. It initially looked like a promising job. I was to join a team working on a Linux-based, "modern" C++ codebase that had been in development for a number of years. Exciting stuff, if you have the same peculiar fetishes as me.

The work wasn't smooth sailing at first, but you never expect an easy ride when you start to work in a new team on a new codebase. However, it didn't get any better as the days (and weeks) rolled by. The code took a *fantastically* long time to learn, and there were no obvious routes into the system. That was a warning sign. At the microlevel, looking at individual lines, methods, and components, the code was messy and badly put together. There was no consistency, no style, and no unifying concepts drawing the separate parts together. That was another warning sign. Control flew around the system in unfathomable and unpredictable ways. That was yet another warning sign. There were so many bad "code smells" (Fowler 1999)