A less obvious benefit is latency. At the construction site there is a second team of carpenters who use the boards to build a house. The boards cannot be used until they are inspected. With the first method, the second team cannot begin its work until all the boards are cut and at least one is inspected. The chances are high that the boards will be delivered in a big batch after they have all been inspected. With the small-batch method, the new boards are delivered without this delay.

The sections that follow relate the small batches principle to system administration and show many benefits beyond reduced waste and improved latency.

2.2 Fixing Hell Month

A company had a team of software developers who produced a new release every six months. When a release shipped, the operations team stopped everything and deployed the release into production. The process took three or four weeks and was very stressful for all involved. Scheduling the maintenance window required complex negotiation. Testing each release was complex and required all hands on deck. The actual software installation never worked on the first try. Once it was deployed, a number of high-priority bugs would be discovered, and each would be fixed by various "hot patches" that would follow.

Even though the deployment process was labor intensive, there was no attempt to automate it. The team had many rationalizations that justified this omission. The production infrastructure changed significantly between releases, making each release a moving target. It was believed that any automation would be useless by the next release because each release's installation instructions were shockingly different. With each next release being so far away, there was always a more important "burning issue" that had to be worked on first. Thus, those who did want to automate the process were told to wait until tomorrow, and tomorrow never came. Lastly, everyone secretly hoped that maybe, just maybe, the next release cycle wouldn't be so bad. Such optimism is a triumph of hope over experience.

Each release was a stressful, painful month for all involved. Soon it was known as Hell Month. To make matters worse, each release was usually late. This made it impossible for the operations team to plan ahead. In particular, it was difficult to schedule any vacation time, which just made everyone more stressed and unhappy.

Feeling compassion for the team's woes, someone proposed that releases should be done less often, perhaps every 9 or 12 months. If something is painful, it is natural to want to do it less frequently.

To everyone's surprise the operations team suggested going in the other direction: monthly releases. This was a big-batch situation. To improve, the company didn't need bigger batches, it needed smaller ones.

People were shocked! Were they proposing that every month be Hell Month? No, by doing it more frequently, there would be pressure to automate the process. If something happens infrequently, there's less urgency to automate it, and we procrastinate. Also, there would be fewer changes to the infrastructure between each release. If an infrastructure change did break the release automation, it would be easier to fix the problem.

The change did not happen overnight. First the developers changed their methodology from mega-releases, with many new features, to small iterations, each with a few specific new features. This was a big change, and selling the idea to the team and management was a long process.

Meanwhile, the operations team automated the testing and deployment processes. The automation could take the latest code, test it, and deploy it into the beta-test area in less than an hour. Pushing code to production was still manual, but by reusing code for the beta rollouts it became increasingly less manual over time.

The result was that the beta area was updated multiple times a day. Since it was automated, there was little reason not to. This made the process continuous, instead of periodic. Each code change triggered the full testing suite, and problems were found in minutes rather than in months.

Pushes to the production area happened monthly because they required coordination among engineering, marketing, sales, customer support, and other groups. That said, all of these teams loved the transition from an unreliable hopefully every-six-months schedule to a reliable monthly schedule. Soon these teams started initiatives to attempt weekly releases, with hopes of moving to daily releases. In the new small-batch world, the following benefits were observed:

- **Features arrived faster.** Where in the past a new feature took up to six months to reach production, now it could go from idea to production in days.
- **Hell Month was eliminated.** After hundreds of trouble-free pushes to beta, pushing to production was easier than ever.
- **The operations team could focus on higher-priority projects.** The operations team was no longer directly involved in software releases other than fixing the automation, which was rare. This freed up the team for more important projects.
- **There were fewer impediments to fixing bugs.** The first step in fixing a bug is to identify which code change is responsible. Big-batch releases had hundreds or thousands of changes to sort through to identify the guilty party. With small batches, it was usually quite obvious where to find the bug.
- **Bugs were fixed in less time.** Fixing a bug in code that was written six months ago is much more difficult than if the code is still fresh in your mind. Small

batches meant bugs were reported soon after the code was written, which meant developers could fix it more expertly in a shorter amount of time.

- **Developers experienced instant gratification.** Waiting six months to see the results of your efforts is demoralizing. Seeing your code help people shortly after it was written is addictive.
- **Everyone was less stressed.** Most importantly, the operations team could finally take long vacations, the kind that require advance planning and scheduling, thus giving them a way to reset and live healthier lives.

While these technical benefits were worthwhile, the business benefits were even more exciting:

- **Improved ability to compete:** Confidence in the ability to add features and fix bugs led to the company becoming more aggressive about new features and fine-tuning existing ones. Customers noticed and sales improved.
- **Fewer missed opportunities:** The sales team had been turning away business due to the company's inability to strike fast and take advantage of opportunities as they arrived. Now the company could enter markets it hadn't previously imagined.
- **A culture of automation and optimization:** Rapid releases removed common excuses not to automate. New automation brought consistency, repeatability, and better error checking, and required less manual labor. Plus, automation could run anytime, not just when the operations team was available.

The ability to do rapid releases is often called a DevOps strategy. In Chapter 20, "Service Launch: DevOps," you'll see similar strategies applied to third-party software.

The Inner and Outer Release Loops

You can think of this process as two nested loops. The inner loop is the code changes done one at a time. The outer loop is the releases that move these changes to production.

2.3 Improving Emergency Failovers

Stack Overflow's main web site infrastructure is in a datacenter in New York City. If the datacenter fails or needs to be taken down for maintenance, duplicate equipment and software are running in Colorado. The duplicate in Colorado is a