In each case, we took the approach of exploring options thoroughly before making decisions. We would make a decision at the "last responsible moment," that point where the cost of *not* deciding outweighed the cost of implementing the feature. Although there were a few things that we might have done differently if Spring had been there from the start, we were not harmed by adding it later. In those early iterations, we focused on uncovering what the application wanted to be rather than how Spring wants us to build applications.

DVD loading

The `DvdLoader` program, which runs in the printing facility, is really a batch processor that reads orders from DVDs and loads them into PCS. As with everything else, we focused on robustness. `DvdLoader` reads an entire order, verifying that the DVD includes all the constituent elements, before it adds the order to PCS. That way it doesn't leave partial or corrupted orders in the database.

Because images can appear on many DVDs, the loader checks to see whether there's already an image loaded with that GUID. If not, the loader adds it. Orders can therefore be resent from the studio whenever necessary, even if PCS has already purged the order and its underlying images. This also means that the background images used in a design get loaded the first time an order for that design arrives.

The DVDs are therefore self-contained and idempotent.

Render pipeline

For the render engine itself, we drew on the classic pipes and filters architecture. "Pipeline" is a natural metaphor for rendering images, and separating the complex sequence of actions into discrete steps also made unit testing simple.

On pulling a job from PCS, the render engine creates a `RenderRequest`. It passes the `RenderRequest` into the rendering pipeline, where each stage operates on the request itself. One of the final stages in the pipeline saves the rendered image to the path specified by PCS. By the time the request exits the pipeline, it holds only a result object with a success indicator and an optional collection of problems.

Each step in the pipeline has its own opportunity to report problems by adding an error message to the result. If any step reports errors, the pipeline aborts and the engine reports the problem back to PCS.

Fail fast

Every system has failure modes; the only question is whether you design them in or just let them happen. We took care to design in "safe" failures, particularly in the production process. There was no way we wanted our software to be responsible for stopping the production line.