runs in the production facility, reading DVDs and calling various stored procedures within PCS to add orders, compositions, and images. PCS treats the composition instructions as an opaque string, and we were careful to avoid any decisions that would have PCS "opening up" the XML in that string. That sometimes means we duplicate information, such as dependencies on the base images themselves, but that is an acceptable trade-off for maintaining a clear boundary.

Similarly, we defined an interface that let the RenderEngine pull render jobs from PCS while keeping the XML description of the rendering itself under Creation Center's control.

We worked out written specifications of those interfaces, and then used FIT running on our development server to "nail down" the precise meaning. In effect, we used FIT as an executable specification of the interfaces. That turned out to be vital because even the people who negotiated the interface still found discrepancies between what they thought they agreed to and what they actually built. FIT let us eliminate those discrepancies during development rather than during integration testing, or worse, in production.

INCREMENTAL ARCHITECTURE

One of the recurring questions in the agile community is, "How much architecture should you create up front?" Some of the leading agile thinkers will tell you, "None. Refactor mercilessly and the architecture will emerge." I've never been in that camp.

Refactoring improves the design of code without changing its functionality. But, to refactor your way to better design, you must first be able to recognize good and bad design. We have a good catalog of "code smells" to guide us there, but I don't know of any equivalent for "architecture smells." Second, it must be possible to change things continuously even across interface boundaries. This has always led me to believe that a system's fundamental architecture must be in place at the start of development.

Now, after the Creation Center project, I'm much less confident in that answer. We added major pieces of the architecture relatively late in the project. Here are some examples:

- Hibernate: Added after two or three iterations. We didn't need the database before this.
- Spring: Added nearly one-third of the way to release 1.0. It quickly became central to our architecture. I don't remember how we got along without it, but we did.
- FIT: Added halfway to release 1.0.
- DVD-burning software: Purchased and added near the end of initial development.
- Support for windowed UIs: Added in the final two iterations before launch.