would lead to one architecture, whereas starting with a business process model might lead to a different architecture. In the extreme case, there is no decomposition, and the system is developed as a monolithic block of software. This might satisfy all functional requirements, but it probably will not satisfy quality concerns such as changeability, maintainability, or scalability. Architects often must do architecture-level refactoring of a system, for example to move from simplex to distributed deployment, or from single-threaded to multithreaded in order to meet scalability or performance requirements, or hardcoded parameters to external configuration files because parameters that were *never* going to change now need to be modified.

Although there are many architectures that can meet functional requirements, only a subset of these will also satisfy quality requirements. Let's go back to the web application example. Think of the many ways to serve up web pages—Apache with static pages, CGI, servlets, JSP, JSF, PHP, Ruby on Rails, or ASP.NET, to name just a few. Choosing one of these technologies is an architecture decision that will have significant impact on your ability to meet certain quality requirements. For example, an approach such as Ruby on Rails might provide the fast time-to-market benefit, but could be harder to maintain as both the Ruby language and the Rails framework continue to evolve rapidly. Or perhaps our application is a web-based telephone and we need to make the phone "ring." If you need to send true asynchronous events from the server to the web page to satisfy performance requirements, an architecture based on servlets might be more testable and modifiable.

In real-world projects, satisfying stakeholder concerns requires many more decisions than simply selecting a web framework. Do you really need an "architecture," and do you need an "architect" to make the decisions? Who should make them? Is it the coder, who may make many of them unintentionally and implicitly, or is it the architect, who makes them explicitly with a view in mind of the entire system, its stakeholders, and its evolution? Either way, you will have an architecture. Should it be explicitly developed and documented, or should it be implicit and require reading of the code to discover?

Often, of course, the choice is not so stark. As the size of the system, its complexity, and the number of people who work on it increase, however, those early decisions and the way that they are documented will have greater and greater impact.

We hope you understand by now that architecture decisions are important if your system is going to meet its quality requirements, and that you want to pay attention to the architecture and make these decisions intentionally rather than just "letting the architecture emerge."

What happens when the system is very large? One of the reasons that we apply architecture principles such as "divide and conquer" is to reduce complexity and enable work to proceed in parallel. This allows us to create larger and larger systems. Can the architecture itself be decomposed into parts, and those parts worked on by different people in parallel? In considering computer architecture, Gerrit Blaauw and Fred Brooks asserted:

> ...if, after all techniques to make the task manageable by a single mind have been applied, the architectural task is still so large and complex that it cannot be done in that way, the product