tem, the more critical is this partitioning—and hence, architecture. And as we will see, the more demanding those quality attributes are, the more critical the architecture is.

A single system is almost inevitably partitioned simultaneously in a number of different ways. Each partitioning results in the creation of an architectural structure: different sets of parts and different relations among the parts. Each is the result of careful design, carried out to satisfy the driving quality attribute requirements and the most important business goals behind the system.

Architecture is what makes the sets of parts work together as a coherent and successful whole. Architecture documentation help architects make the right decisions; it tells developers how to carry them out; and it records those decisions to give a system's future caretakers insight into the architect's solution.

Many projects make the mistake of trying to impose a single partition in multiple component domains, such as equating threads with objects, which are equated with modules, which in turn are equated with files. Such an approach never succeeds fully, and adjustments eventually must be made, but the damage of the initial intent is often hard to repair. This invariably leads to problems in development and occasionally in final products.

—Jazayeri, Ran, and van der Linden (2000, pp. 16–17)

P.1.2 Architecture and Quality Attributes

For nearly all systems, quality attributes such as performance, reliability, security, and modifiability are every bit as important as making sure that the software computes the correct answer. A software system's ability to produce correct results isn't helpful if it takes too long doing it, or the system doesn't stay up long enough to deliver it, or the system reveals the results to your competition or your enemy. Architecture is where these concerns are addressed. For example:

- If you require high performance, you need to
  - Exploit potential parallelism by decomposing the work into cooperating or synchronizing processes.
  - Manage the interprocess and network communication volume and data access frequencies.
  - Be able to estimate expected latencies and throughputs.
  - Identify potential performance bottlenecks.
- If your system needs high accuracy, you must pay attention to how the data elements are defined and used and how their values flow throughout the system.
- If security is important, you need to
  - Legislate usage relationships and communication restrictions among the parts.
  - Identify parts of the system where an unauthorized intrusion will do the most damage.
  - Possibly introduce special elements that have earned a high degree of trust.

- If you need to support modifiability and portability, you must carefully separate concerns among the parts of the system, so that when a change affects one element, that change does not ripple across the system.
- If you want to deploy the system incrementally, by releasing successively larger subsets, you have to keep the dependency relationships among the pieces untangled, to avoid the "nothing works until everything works" syndrome.

The solutions to these concerns are purely architectural in nature. It is up to architects to find those solutions and communicate them effectively to those who will carry them out. Architecture documentation has three obligations related to quality attributes. First, it should indicate which quality attribute requirements drove the design. Second, it should capture the solutions chosen to satisfy the quality attribute requirements. Finally, it should capture a convincing argument why the solutions provide the necessary quality attributes. The goal is to capture enough information so that the architecture can be analyzed to see if, in fact, the system(s) derived from it will possess the necessary quality attributes.

Chapter 10 will show where in the documentation to record the driving quality attribute requirements, the solutions chosen, and the rationale for those solutions.

COMING TO TERMS

What Is Software Architecture?

If we are to agree on what it means to document a software architecture, we should establish a common basis for what it is we're documenting. No universal definition of software architecture exists. The Software Engineering Institute's Web site collects definitions from the literature and from practitioners around the world; so far, more than 150 definitions have been collected.

It seems that new fields try to nail down standard definitions or their key terms as soon as they can. As the field matures, basic concepts become more important than ironclad definitions, and this urge seems to fade. When object-oriented development was in its infancy, you could bring any OO meeting to a screeching halt by putting on your best innocent face and asking, "What exactly is an object?" This largely ended when people realized that the scatter plot of definitions had an apparent (if unarticulated) centroid, from which very useful progress could be made. Sometimes "close enough" is, well, close enough.

Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled.

—Eoin Woods (SEI 2010)

You can read the SEI collection of definitions, or contribute your own, at www.sei.cmu.edu/architecture.