2

Evaluating a Software Architecture

> Marry your architecture in haste and you can repent in leisure.
>
> —Barry Boehm
>
> from a keynote address: *And Very Few Lead Bullets Either*

How can you be sure whether the architecture chosen for your software is the right one? How can you be sure that it won't lead to calamity but instead will pave the way through a smooth development and successful product?

It's not an easy question, and a lot rides on the outcome. The foundation for any software system is its architecture. The architecture will allow or preclude just about all of a system's quality attributes. Modifiability, performance, security, availability, reliability—all of these are precast once the architecture is laid down. No amount of tuning or clever implementation tricks will wring any of these qualities out of a poorly architected system.

To put it bluntly, an architecture is a bet, a wager on the success of a system. Wouldn't it be nice to know in advance if you've placed your bet on a winner, as opposed to waiting until the system is mostly completed before knowing whether it will meet its requirements or not? If you're buying a system or paying for its development, wouldn't you like to have some assurance that it's started off down the right path? If you're the architect yourself, wouldn't you like to have a good way to validate your intuitions and experience, so that you can sleep at night knowing that the trust placed in your design is well founded?

Until recently, there were almost no methods of general utility to validate a software architecture. If performed at all, the approaches were spotty, ad hoc, and not repeatable. Because of that, they weren't particularly trustworthy. We can do better than that.

This is a guidebook of software architecture evaluation. It is built around a suite of three methods, all developed at the Software Engineering Institute, that can be applied to any software-intensive system:

- ATAM: Architecture Tradeoff Analysis Method
- SAAM: Software Architecture Analysis Method
- ARID: Active Reviews for Intermediate Designs

The methods as a group have a solid pedigree, having been applied for years on dozens of projects of all sizes and in a wide variety of domains. With these methods, the time has come to include software architecture evaluation as a standard step of any development paradigm. Evaluations represent a wise risk-mitigation effort and are relatively inexpensive. They pay for themselves in terms of costly errors and sleepless nights avoided.

Whereas the previous chapter introduced the concept of software architecture, this chapter lays the conceptual groundwork for architectural evaluation. It defines what we mean by software architecture and explains the kinds of properties for which an architecture can (and cannot) be evaluated.

First, let's restate what it is we're evaluating:

> *The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them. [Bass 98]*

By "externally visible" properties, we are referring to those assumptions other components can make of a component, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. The intent of this definition is that a software architecture must abstract some information about the system (otherwise there is no point looking at the architecture—we are simply viewing the entire system) and yet provide enough information to be a basis for analysis, decision making, and hence risk reduction (see the sidebar What's Architectural?).

The architecture defines the components (such as modules, objects, processes, subsystems, compilation units, and so forth) and the relevant relations (such as calls, sends-data-to, synchronizes-with, uses, depends-on, instantiates, and many more) among them. The architecture is the result of early design decisions that are necessary before a group of people can collaboratively build a software system. The larger or more distributed the group, the more vital the architecture is (and the group doesn't have to be very large before the architecture is vital).

One of the insights about architecture from Chapter 1 that you must fully embrace before you can understand architecture evaluation is this:

> *Architectures allow or preclude nearly all of the system's quality attributes.*

This leads to the most fundamental truth about architecture evaluation: If architectural decisions determine a system's quality attributes, then it is possible to evaluate architectural decisions with respect to their impact on those attributes.

What's Architectural?

Sooner or later everyone asks the question: "What's architectural?" Some people ask out of intellectual curiosity, but people who are evaluating architectures have a pressing need to understand what information is in and out of their realm of concern. Maybe you didn't ask the question exactly that way. Perhaps you asked it in one of the following ways:

- What is the difference between an architecture and a high-level design?
- Are details such as priorities of processes architectural?
- Why should implementation considerations such as buffer overflows be treated as architectural?
- Are interfaces to components part of the architecture?
- If I have class diagrams, do I need anything else?
- Is architecture concerned with run-time behavior or static structure?
- Is the operating system part of the architecture? Is the programming language?
- If I'm constrained to use a particular commercial product, is that architectural? If I'm free to choose from a wide range of commercial products, is that architectural?

Let's think about this in two ways.

First, consider the definition of architecture that we quoted in Chapter 1 of this book. Paraphrasing: A software architecture concerns the gross organization of a system described in terms of its components, their externally visible properties, and the relationships among them. True enough, but it fails to explicitly address the notion of context. If the scope of my concern is confined to a subsystem within a system that is part of a system of systems, then what I consider to be architectural will be different than what the architect of the system of systems considers to be architectural. Therefore, context influences what's architectural.

Second, let's ask, what is *not* architectural? It has been said that algorithms are not architectural; data structures are not architectural; details of data flow are not architectural. Well, again these statements are only partially true. Some properties of algorithms, such as their complexity, might have a dramatic effect on performance. Some properties of data structures, such as

whether they need to support concurrent access, directly impact performance and reliability. Some of the details of data flow, such as how components depend on specific message types or which components are allowed access to which data types, impact modifiability and security, respectively.

So is there a principle that we can use in determining what is architectural? Let's appeal to what architecture is used for to formulate our principle. Our criterion for something to be architectural is this: It must be a component, or a relationship between components, or a property (of components or relationships) *that needs to be externally visible* in order to reason about the ability of the system to meet its quality requirements or to support decomposition of the system into independently implementable pieces. Here are some corollaries of this principle:

- *Architecture describes what is in your system.* When you have determined your context, you have determined a boundary that describes what is in and what is out of your system (which might be someone else's subsystem). Architecture describes the part that is in.
- *An architecture is an abstract depiction of your system.* The information in an architecture is the most abstract and yet meaningful depiction of that aspect of the system. Given your architectural specification, there should not be a need for a more abstract description. That is not to say that all aspects of architecture are abstract, nor is it to say that there is an abstraction threshold that needs to be exceeded before a piece of design information can be considered architectural. You shouldn't worry if your architecture encroaches on what others might consider to be a more detailed design.
- *What's architectural should be critical for reasoning about critical requirements.* The architecture bridges the gap between requirements and the rest of the design. If you feel that some information is critical for reasoning about how your system will meet its requirements then it is architectural. You, as the architect, are the best judge. On the other hand, if you can eliminate some details and still compose a forceful argument through models, simulation, walk-throughs, and so on about how your architecture will satisfy key requirements then those details do not belong. However, if you put too much detail into your architecture then it might not satisfy the next principle.
- *An architectural specification needs to be graspable.* The whole point of a gross-level system depiction is that you can understand it and reason about it. Too much detail will defeat this purpose.
- *An architecture is constraining.* It imposes requirements on all lower-level design specifications. I like to distinguish between when a decision is made and when it is realized. For example, I might determine a process prioritization strategy, a component redundancy strategy, or a set of encapsulation rules when designing an architecture; but I might not actually make priority assignments, determine the algorithm for a redundant calculation, or specify the details of an interface until much later.

In a nutshell:

To be architectural is to be the most abstract depiction of the system that enables reasoning about critical requirements and constrains all subsequent refinements.

If it sounds like finding all those aspects of your system that are architectural is difficult, that is true. It is unlikely that you will discover everything that is architectural up front, nor should you try. An architectural specification will evolve over time as you continually apply these principles in determining what's architectural.

—MHK

2.1 Why Evaluate an Architecture?

The earlier you find a problem in a software project, the better off you are. The cost to fix an error found during requirements or early design phases is orders of magnitudes less to correct than the same error found during testing. Architecture is the product of the early design phase, and its effect on the system and the project is profound.

An unsuitable architecture will precipitate disaster on a project. Performance goals will not be met. Security goals will fall by the wayside. The customer will grow impatient because the right functionality is not available, and the system is too hard to change to add it. Schedules and budgets will be blown out of the water as the team scrambles to back-fit and hack their way through the problems. Months or years later, changes that could have been anticipated and planned for will be rejected because they are too costly. Plagues and pestilence cannot be too far behind.

Architecture also determines the structure of the project: configuration control libraries, schedules and budgets, performance goals, team structure, documentation organization, and testing and maintenance activities all are organized around the architecture. If it changes midstream because of some deficiency discovered late, the entire project can be thrown into chaos. It is much better to change the architecture before it has been frozen into existence by the establishment of downstream artifacts based on it.

Architecture evaluation is a cheap way to avoid disaster. The methods in this book are meant to be applied while the architecture is a paper specification (of course, they can be applied later as well), and so they involve running a series of simple thought experiments. They each require assembling relevant stakeholders for a structured session of brainstorming, presentation, and analysis. All told, the average architecture evaluation adds no more than a few days to the project schedule.

To put it another way, if you were building a house, you wouldn't think of proceeding without carefully looking at the blueprints before construction began. You would happily spend the small amount of extra time because you know it's much better to discover a missing bedroom while the architecture is just a blueprint, rather than on moving day.

2.2 When Can an Architecture Be Evaluated?

The classical application of architecture evaluation occurs when the architecture has been specified but before implementation has begun. Users of iterative or incremental life-cycle models can evaluate the architectural decisions made during the most recent cycle. However, one of the appealing aspects of architecture evaluation is that it can be applied at any stage of an architecture's lifetime, and there are two useful variations from the classical: early and late.

Early. Evaluation need not wait until an architecture is fully specified. It can be used at any stage in the architecture creation process to examine those architectural decisions already made and choose among architectural options that are pending. That is, it is equally adept at evaluating architectural decisions that have already been made and those that are being considered.

Of course, the completeness and fidelity of the evaluation will be a direct function of the completeness and fidelity of the architectural description brought to the table by the architect. And in practice, the expense and logistical burden of convening a full-blown evaluation is seldom undertaken when unwarranted by the state of the architecture. It is just not going to be very rewarding to assemble a dozen or two stakeholders and analysts to evaluate the architect's early back-of-the-napkin sketches, even though such sketches will in fact reveal a number of significant architecture paths chosen and paths not taken.

Some organizations recommend what they call a *discovery review*, which is a very early mini-evaluation whose purpose is as much to iron out and prioritize troublesome requirements as analyzing whatever "proto-architecture" may have been crafted by that point. For a discovery review, the stakeholder group is smaller but must include people empowered to make requirements decisions. The purpose of this meeting is to raise any concerns that the architect may have about the feasibility of *any* architecture to meet the combined quality and behavioral requirements that are being levied while there is still time to relax the most troubling or least important ones. The output of a discovery review is a much stronger set of requirements and an initial approach to satisfying them. That approach, when fleshed out, can be the subject of a full evaluation later.

We do not cover discovery reviews in detail because they are a straightforward variation of an architecture evaluation. If you hold a discovery review, make sure to

- Hold it before the requirements are frozen and when the architect has a good idea about how to approach the problem
- Include in the stakeholder group someone empowered to make requirements decisions
- Include a prioritized set of requirements in the output, in case there is no apparent way to meet all of them

Finally, in a discovery review, remember the words of the gifted aircraft designer Willy Messerschmitt, himself no stranger to the burden of requirements, who said:

> *You can have any combination of features the Air Ministry desires, so long as you do not also require that the resulting airplane fly.*

Late. The second variation takes place when not only the architecture is nailed down but the implementation is complete as well. This case occurs when an organization inherits some sort of legacy system. Perhaps it has been purchased on the open market, or perhaps it is being excavated from the organization's own archives. The techniques for evaluating a legacy architecture are the same as those for one that is newborn. An evaluation is a useful thing to do because it will help the new owners understand the legacy system, and let them know whether the system can be counted on to meet its quality and behavioral requirements.

In general, when can an architectural evaluation be held? As soon as there is enough of an architecture to justify it. Different organizations may measure that justification differently, but a good rule of thumb is this: Hold an evaluation when development teams start to make decisions that depend on the architecture and the cost of undoing those decisions would outweigh the cost of holding an evaluation.

2.3 Who's Involved?

There are two groups of people involved in an architecture evaluation.

1. *Evaluation team.* These are the people who will conduct the evaluation and perform the analysis. The team members and their precise roles will be defined later, but for now simply realize that they represent one of the classes of participants.
2. *Stakeholders.* Stakeholders are people who have a vested interest in the architecture and the system that will be built from it. The three evaluation methods in this book all use stakeholders to articulate the specific requirements that are levied on the architecture, above and beyond the requirements that state what functionality the system is supposed to exhibit. Some, but not all, of the stakeholders will be members of the development team: coders, integrators, testers, maintainers, and so forth.

   A special kind of stakeholder is a project decision maker. These are people who are interested in the outcome of the evaluation and have the power to make decisions that affect the future of the project. They include the architect, the designers of components, and the project's management. Management will have to make decisions about how to respond to the issues raised by the evaluation. In some settings (particularly government acquisitions), the customer or sponsor may be a project decision maker as well.

   Whereas an arbitrary stakeholder says what he or she wants to be true about the architecture, a decision maker has the power to expend resources to *make* it true. So a project manager might say (as a stakeholder), "I would like the architecture to be reusable on a related project that I'm managing," but as a decision maker he or she might say, "I see that the changes you've identified as necessary to reuse this architecture on my other project are too expensive, and I won't pay for them." Another difference is that a project decision maker has the power to speak authoritatively for the project, and some of the steps of the ATAM method, for example, ask them to do precisely that. A garden-variety stakeholder, on the other hand, can only hope to influence (but not control) the project. For more on stakeholders, see the sidebar Stakeholders on page 63 in Chapter 3.

The client for an architecture evaluation will usually be a project decision maker, with a vested interest in the outcome of the evaluation and holding some power over the project.

Sometimes the evaluation team is drawn from the project staff, in which case they are also stakeholders. This is not recommended because they will lack the objectivity to view the architecture in a dispassionate way.

2.4 What Result Does an Architecture Evaluation Produce?

In concrete terms, an architecture evaluation produces a report, the form and content of which vary according to the method used. Primarily, though, an architecture evaluation produces information. In particular, it produces answers to two kinds of questions.

1. Is this architecture suitable for the system for which it was designed?
2. Which of two or more competing architectures is the most suitable one for the system at hand?

Suitability for a given task, then, is what we seek to investigate. We say that an architecture is suitable if it meets two criteria.

1. The system that results from it will meet its quality goals. That is, the system will run predictably and fast enough to meet its performance (timing) requirements. It will be modifiable in planned ways. It will meet its security constraints. It will provide the required behavioral function. Not every quality property of a system is a direct result of its architecture, but many are, and for those that are, the architecture is suitable if it provides the blueprint for building a system that achieves those properties.
2. The system can be built using the resources at hand: the staff, the budget, the legacy software (if any), and the time allotted before delivery. That is, the architecture is *buildable*.

This concept of suitability will set the stage for all of the material that follows. It has a couple of important implications. First, suitability is only relevant in the context of specific (and specifically articulated) goals for the architecture and the system it spawns. An architecture designed with high-speed performance as the primary design goal might lead to a system that runs like the wind but requires hordes of programmers working for months to make any kind of modification to it. If modifiability were more important than performance *for that system*, then that architecture would be unsuitable *for that system* (but might be just the ticket for another one).

In *Alice in Wonderland,* Alice encounters the Cheshire Cat and asks for directions. The cat responds that it depends upon where she wishes to go. Alice says she doesn't know, whereupon the cat tells her it doesn't matter which way she walks. So

> *If the sponsor of a system cannot tell you what any of the quality goals are for the system, then any architecture will do.*

An overarching part of an architecture evaluation is to capture and prioritize specific goals that the architecture must meet in order to be considered

Why Should I Believe You?

Frequently when we embark on an evaluation we are outsiders. We have been called in by a project leader or a manager or a customer to evaluate a project. Perhaps this is seen as an audit, or perhaps it is just part of an attempt to improve an organization's software engineering practice. Whatever the reason, unless the evaluation is part of a long-term relationship, we typically don't personally know the architect, or we don't know the major stakeholders.

Sometimes this distance is not a problem—the stakeholders are receptive and enthusiastic, eager to learn and to improve their architecture. But on other occasions we meet with resistance and perhaps even fear. The major players sit there with their arms folded across their chests, clearly annoyed that they have been taken away from their *real* work, that of architecting, to pursue this silly management-directed evaluation. At other times the stakeholders are friendly and even receptive, but they are skeptical. After all, they are the experts in their domains and they have been working in the area, and maybe even on this system, for years.

In either case their attitudes, whether friendly or unfriendly, indicate a substantial amount of skepticism over the prospect that the evaluation can actually help. They are in effect saying, "What could a bunch of outsiders possibly have to tell us about *our* system that we don't already know?" You will probably have to face this kind of opposition or resistance at some point in your tenure as an architecture evaluator.

There are two things that you need to know and do to counteract this opposition. First of all, you need to counteract the fear. So keep calm. If you are friendly and let them know that the point of the meeting is to learn about and improve the architecture (rather than pointing a finger of blame) then you will find that resistance melts away quickly. Most people actually enjoy the evaluation process and see the benefits very quickly. Second, you need to counteract the skepticism. Of course they are the experts in the domain. You know this and they know this, and you should acknowledge this up front. But you are the architecture and quality attribute expert. No matter what the domain, architectural approaches for dealing with and analyzing quality attributes don't vary much. There are relatively few ways to approach performance or availability or security on an architectural level. As an experienced evaluator (and with the help of the insight from the quality attribute communities) you have seen these before, and they don't change much from domain to domain.

Furthermore, as an outsider you bring a "fresh set of eyes," and this alone can often bring new insights into a project. Finally, you are following a process that has been refined over dozens of evaluations covering dozens of different domains. It has been refined to make use of the expertise of many people, to elicit, document, and cross-check quality attribute requirements and architectural information. This alone will bring benefit to your project—we have seen it over and over again. The process works!

—*RK*

suitable. In a perfect world, these would all be captured in a requirements document, but this notion fails for two reasons: (1) Complete and up-to-date requirements documents don't always exist, and (2) requirements documents express the requirements for a *system*. There are additional requirements levied on an architecture besides just enabling the system's requirements to be met. (Buildability is an example.)

The second implication of evaluating for suitability is that the answer that comes out of the evaluation is not going to be the sort of scalar result you may be used to when evaluating other kinds of software artifacts. Unlike code metrics, for example, in which the answer might be 7.2 and anything over 6.5 is deemed unacceptable, an architecture evaluation is going to produce a more thoughtful result.

We are not interested in precisely characterizing any quality attribute (using measures such as mean time to failure or end-to-end average latency). That would be pointless at an early stage of design because the actual parameters that determine these values (such as the actual execution time of a component) are often implementation dependent. What we are interested in doing—in the spirit of a risk-mitigation activity—is learning where an attribute of interest is affected by architectural design decisions, so that we can reason carefully about those decisions, model them more completely in subsequent analyses, and devote more of our design, analysis, and prototyping energies to such decisions.

An architectural evaluation will tell you that the architecture has been found suitable with respect to one set of goals and problematic with respect to another set of goals. Sometimes the goals will be in conflict with each other, or at the very least, some goals will be more important than other ones. And so the manager of the project will have a decision to make if the architecture evaluates well in some areas and not so well in others. Can the manager live with the areas of weakness? Can the architecture be strengthened in those areas? Or is it time for a wholesale restart? The evaluation will help reveal where an architecture is weak, but weighing the cost against benefit to the project of strengthening the architecture is solely a function of project context and is in the realm of management. So

> *An architecture evaluation doesn't tell you "yes" or "no," "good" or "bad," or "6.75 out of 10." It tells you where you are at risk.*

Architecture evaluation can be applied to a single architecture or to a group of competing architectures. In the latter case, it can reveal the strengths and weaknesses of each one. Of course, you can bet that no architecture will evaluate better than all others in all areas. Instead, one will outperform others in some areas but underperform in other areas. The evaluation will first identify what the areas of interest are and then highlight the strengths and weaknesses of each architecture in those areas. Management must decide which (if any) of

the competing architectures should be selected or improved or whether none of the candidates is acceptable and a new architecture should be designed.[1]

2.5 For What Qualities Can We Evaluate an Architecture?

In this section, we say more precisely what suitability means. It isn't quite true that we can tell from looking at an architecture whether the ensuing system will meet *all* of its quality goals. For one thing, an implementation might diverge from the architectural plan in ways that subvert the quality plans. But for another, architecture does not strictly determine all of a system's qualities.

Usability is a good example. Usability is the measure of a user's ability to utilize a system effectively. Usability is an important quality goal for many systems, but usability is largely a function of the user interface. In modern systems design, particular aspects of the user interface tend to be encapsulated within small areas of the architecture. Getting data to and from the user interface and making it flow around the system so that the necessary work is done to support the user is certainly an architectural issue, as is the ability to change the user interface should that be required. However, many aspects of the user interface—whether the user sees red or blue backgrounds, a radio button or a dialog box—are by and large not architectural since those decisions are generally confined to a limited area of the system.

But other quality attributes lie squarely in the realm of architecture. For instance, the ATAM concentrates on evaluating an architecture for suitability in terms of imbuing a system with the following quality attributes. (Definitions are based on Bass et al. [Bass 98])

- *Performance:* Performance refers to the responsiveness of the system—the time required to respond to stimuli (events) or the number of events processed in some interval of time. Performance qualities are often expressed by the number of transactions per unit time or by the amount of time it takes to complete a transaction with the system. Performance measures are often cited using *benchmarks*, which are specific transaction sets or workload conditions under which the performance is measured.
- *Reliability:* Reliability is the ability of the system to keep operating over time. Reliability is usually measured by mean time to failure.

1. This is the last time we will address evaluating more than one architecture at a time since the methods we describe are carried out in the same fashion for either case.

- *Availability:* Availability is the proportion of time the system is up and running. It is measured by the length of time between failures as well as how quickly the system is able to resume operation in the event of failure.
- *Security:* Security is a measure of the system's ability to resist unauthorized attempts at usage and denial of service while still providing its services to legitimate users. Security is categorized in terms of the types of threats that might be made to the system.
- *Modifiability:* Modifiability is the ability to make changes to a system quickly and cost effectively. It is measured by using specific changes as benchmarks and recording how expensive those changes are to make.
- *Portability:* Portability is the ability of the system to run under different computing environments. These environments can be hardware, software, or a combination of the two. A system is portable to the extent that all of the assumptions about any *particular* computing environment are confined to one component (or at worst, a small number of easily changed components). If porting to a new system requires change, then portability is simply a special kind of modifiability.
- *Functionality:* Functionality is the ability of the system to do the work for which it was intended. Performing a task requires that many or most of the system's components work in a coordinated manner to complete the job.
- *Variability:* Variability is how well the architecture can be expanded or modified to produce new architectures that differ in specific, preplanned ways. Variability mechanisms may be run-time (such as negotiating on the fly protocols), compile-time (such as setting compilation parameters to bind certain variables), build-time (such as including or excluding various components or choosing different versions of a component), or code-time mechanisms (such as coding a device driver for a new device). Variability is important when the architecture is going to serve as the foundation for a whole family of related products, as in a product line.
- *Subsetability:* This is the ability to support the production of a subset of the system. While this may seem like an odd property of an architecture, it is actually one of the most useful and most overlooked. Subsetability can spell the difference between being able to deliver nothing when schedules slip versus being able to deliver a substantial part of the product. Subsetability also enables incremental development, a powerful development paradigm in which a minimal system is made to run early on and functions are added to it over time until the whole system is ready. Subsetability is a special kind of variability, mentioned above.
- *Conceptual integrity:* Conceptual integrity is the underlying theme or vision that unifies the design of the system at all levels. The architecture should do similar things in similar ways. Conceptual integrity is exemplified in an architecture that exhibits consistency, has a small number of data

and control mechanisms, and uses a small number of patterns throughout to get the job done.

By contrast, the SAAM concentrates on modifiability in its various forms (such as portability, subsetability, and variability) and functionality. The ARID method provides insights about the suitability of a portion of the architecture to be used by developers to complete their tasks.

If some other quality than the ones mentioned above is important to you, the methods still apply. The ATAM, for example, is structured in steps, some of which are dependent upon the quality being investigated, and others of which are not. Early steps of the ATAM allow you to define new quality attributes by explicitly describing the properties of interest. The ATAM can easily accommodate new quality-dependent analysis. When we introduce the method, you'll see where to do this. For now, though, the qualities in the list above form the basis for the methods' capabilities, and they also cover most of what people tend to be concerned about when evaluating an architecture.

2.6 Why Are Quality Attributes Too Vague for Analysis?

Quality attributes form the basis for architectural evaluation, but simply naming the attributes by themselves is not a sufficient basis on which to judge an architecture for suitability. Often, requirements statements like the following are written:

- "The system shall be robust."
- "The system shall be highly modifiable."
- "The system shall be secure from unauthorized break-in."
- "The system shall exhibit acceptable performance."

Without elaboration, each of these statements is subject to interpretation and misunderstanding. What you might think of as robust, your customer might consider barely adequate—or vice versa. Perhaps the system can easily adopt a new database but cannot adapt to a new operating system. Is that system maintainable or not? Perhaps the system uses passwords for security, which prevents a whole class of unauthorized users from breaking in, but has no virus protection mechanisms. Is that system secure from intrusion or not?

The point here is that quality attributes are not absolute quantities; they exist in the context of specific goals. In particular:

- A system is modifiable (or not) with respect to a specific kind of change.
- A system is secure (or not) with respect to a specific kind of threat.
- A system is reliable (or not) with respect to a specific kind of fault occurrence.
- A system performs well (or not) with respect to specific performance criteria.
- A system is suitable (or not) for a product line with respect to a specific set or range of envisioned products in the product line (that is, with respect to a specific product line *scope*).
- An architecture is buildable (or not) with respect to specific time and budget constraints.

If this doesn't seem reasonable, consider that no system can ever be, for example, completely reliable under all circumstances. (Think power failure, tornado, or disgruntled system operator with a sledgehammer.) Given that, it is incumbent upon the architect to understand under exactly what circumstances the system should be reliable in order to be deemed acceptable.

In a perfect world, the quality requirements for a system would be completely and unambiguously specified in a requirements document. Most of us do not live in such a world. Requirements documents are not written, or are written poorly, or are not finished when it is time to begin the architecture. Also, architectures have goals of their own that are not enumerated in a requirements document for the system: They must be built using resources at hand, they should exhibit conceptual integrity, and so on. And so the first job of an architecture evaluation is to elicit the specific quality goals against which the architecture will be judged.

If all of these goals are specifically, unambiguously articulated, that's wonderful. Otherwise, we ask the stakeholders to help us write them down during an evaluation. The mechanism we use is the *scenario*. A scenario is a short statement describing an interaction of one of the stakeholders with the system. A user would describe using the system to perform some task; these scenarios would very much resemble *use cases* in object-oriented parlance. A maintenance stakeholder would describe making a change to the system, such as upgrading the operating system in a particular way or adding a specific new function. A developer's scenario might involve using the architecture to build the system or predict its performance. A customer's scenario might describe the architecture reused for a second product in a product line or might assert that the system is buildable given certain resources.

Each scenario, then, is associated with a particular stakeholder (although different stakeholders might well be interested in the same scenario). Each scenario also addresses a particular quality, but in specific terms. Scenarios are discussed more fully in Chapter 3.

2.7 What Are the Outputs of an Architecture Evaluation?

2.7.1 Outputs from the ATAM, the SAAM, and ARID

An architecture evaluation results in information and insights about the architecture. The ATAM, the SAAM, and the ARID method all produce the outputs described below.

Prioritized Statement of Quality Attribute Requirements

An architecture evaluation can proceed only if the criteria for suitability are known. Thus, elicitation of quality attribute requirements against which the architecture is evaluated constitutes a major portion of the work. But no architecture can meet an unbounded list of quality attributes, and so the methods use a consensus-based prioritization. Having a prioritized statement of the quality attributes serves as an excellent documentation record to accompany any architecture and guide it through its evolution. All three methods produce this in the form of a set of quality attribute scenarios.

Mapping of Approaches to Quality Attributes

The answers to the analysis questions produce a mapping that shows how the architectural approaches achieve (or fail to achieve) the desired quality attributes. This mapping makes a splendid rationale for the architecture. Rationale is something that every architect should record, and most wish they had time to construct. The mapping of approaches to attributes can constitute the bulk of such a description.

Risks and Nonrisks

Risks are potentially problematic architectural decisions. Nonrisks are good decisions that rely on assumptions that are frequently implicit in the architecture. Both should be understood and explicitly recorded.[2]

Documenting of risks and nonrisks consists of

- An architectural decision (or a decision that has not been made)
- A specific quality attribute response that is being addressed by that decision along with the consequences of the predicted level of the response

2. Risks can also emerge from other, nonarchitectural sources. For example, having a management structure that is misaligned with the architectural structure might present an organizational risk. Insufficient communication between the stakeholder groups and the architect is a common kind of management risk.

- A rationale for the positive or negative effect that decision has on meeting the quality attribute requirement

  An example of a risk is

  The rules for writing business logic modules in the second tier of your three-tier client-server style are not clearly articulated (*a decision that has not been made*). This could result in replication of functionality, thereby compromising modifiability of the third tier (*a quality attribute response and its consequences*). Unarticulated rules for writing the business logic can result in unintended and undesired coupling of components (*rationale for the negative effect*).

  An example of a nonrisk is

  Assuming message arrival rates of once per second, a processing time of less than 30 milliseconds, and the existence of one higher priority process (*the architectural decisions*), a one-second soft deadline seems reasonable (*the quality attribute response and its consequences*) since the arrival rate is bounded and the preemptive effects of higher priority processes are known and can be accommodated (*the rationale*).

For a nonrisk to remain a nonrisk the assumptions must not change (or at least if they change, the designation of nonrisk will need to be rejustified). For example, if the message arrival rate, the processing time, or the number of higher priority processes changes in the example above, the designation of nonrisk could change.

2.7.2 Outputs Only from the ATAM

In addition to the preceding information, the ATAM produces an additional set of results described below.

Catalog of Architectural Approaches Used

Every architect adopts certain design strategies and approaches to solve the problems at hand. Sometimes these approaches are well known and part of the common knowledge of the field; sometimes they are unique and innovative to the system being built. In either case, they are the key to understanding whether the architecture will meet its goals and requirements. The ATAM includes a step in which the approaches used are catalogued, and this catalog can later serve as an introduction to the architecture for people who need to familiarize themselves with it, such as future architects and maintainers for the system.

Approach- and Quality-Attribute-Specific Analysis Questions

The ATAM poses analysis questions that are based on the attributes being sought and the approaches selected by the architect. As the architecture evolves, these questions can be used in future mini-evaluations to make sure that the evolution is not taking the architecture in the wrong direction.

Sensitivity Points and Tradeoff Points

We term key architectural decisions *sensitivity points* and *tradeoff points*. A sensitivity point is an architectural decision involving one or more architectural components (and/or component relationships) that is critical for achieving a particular quality attribute response measure. We call it this because the response measure is sensitive to changing the decision. For example:

- The level of confidentiality in a virtual private network might be sensitive to the number of bits of encryption.
- The latency for processing an important message might be sensitive to the priority of the lowest priority process involved in handling the message.
- The average number of person-days of effort it takes to maintain a system might be sensitive to the degree of encapsulation of its communication protocols and file formats.

Sensitivity points tell a designer or analyst where to focus attention when trying to understand the achievement of a quality goal. They serve as yellow flags: "Use caution when changing this property of the architecture." Particular values of sensitivity points may become risks when realized in an architecture. Consider the examples above. A particular value in the encryption level—say, 32-bit encryption—may present a risk in the architecture. Or having a very low priority process in a pipeline that processes an important message may become a risk in the architecture.

A *tradeoff point* is an architectural decision that affects more than one attribute and is a sensitivity point for more than one attribute. For example, changing the level of encryption could have a significant impact on both security and performance. Increasing the level of encryption improves the predicted security but requires more processing time. If the processing of a confidential message has a hard real-time latency requirement then the level of encryption could be a tradeoff point. Tradeoff points are the most critical decisions that one can make in an architecture, which is why we focus on them so carefully.

Finally, it is not uncommon for an architect to answer an elicitation question by saying, "We haven't made that decision yet." In this case you cannot point to a component or property in the architecture and call it out as a sensitivity point because the component or property might not exist yet. However, it is important to flag key decisions that have been made as well as key decisions that have not yet been made.

2.8 What Are the Benefits and Costs of Performing an Architecture Evaluation?

The main, and obvious, benefit of architecture evaluation is, of course, that it uncovers problems that if left undiscovered would be orders of magnitude more expensive to correct later. In short, architecture evaluation produces better architectures. Even if the evaluation uncovers no problems that warrant attention, it will increase everyone's level of confidence in the architecture.

But there are other benefits as well. Some of them are hard to measure, but they all contribute to a successful project and a more mature organization. You may not experience all of these on every evaluation, but the following is a list of the benefits we've often observed.

Puts Stakeholders in the Same Room

An architecture evaluation is often the first time that many of the stakeholders have ever met each other; sometimes it's the first time the architect has met them. A group dynamic emerges in which stakeholders see each other as all wanting the same thing: a successful system. Whereas before, their goals may have been in conflict with each other (and in fact, still may be), now they are able to explain their goals and motivations so that they begin to understand each other. In this atmosphere, compromises can be brokered or innovative solutions proposed in the face of greater understanding. It is almost always the case that stakeholders trade phone numbers and e-mail addresses and open channels of communication that last beyond the evaluation itself.

Forces an Articulation of Specific Quality Goals

The role of the stakeholders is to articulate the quality goals that the architecture should meet in order to be deemed successful. These goals are often not captured in any requirements document, or at least not captured in an unambiguous fashion beyond vague platitudes about reliability and modifiability. Scenarios provide explicit quality benchmarks.

Results in the Prioritization of Conflicting Goals

Conflicts that might arise among the goals expressed by the different stakeholders will be aired. Each method includes a step in which the goals are prioritized by the group. If the architect cannot satisfy all of the conflicting goals, he or she will receive clear and explicit guidance about which ones are considered most important. (Of course, project management can step in and veto or adjust the group-derived priorities—perhaps they perceive some stakeholders and their goals as "more equal" than others—but not unless the conflicting goals are aired.)

Forces a Clear Explication of the Architecture

The architect is compelled to make a group of people not privy to the architecture's creation understand it, in detail, in an unambiguous way. Among other things, this will serve as a dress rehearsal for explaining it to the other designers, component developers, and testers. The project benefits by forcing this explication early.

Improves the Quality of Architectural Documentation

Often, an evaluation will call for documentation that has not yet been prepared. For example, an inquiry along performance lines will reveal the need for documentation that shows how the architecture handles the interaction of run-time tasks or processes. If the evaluation requires it, then it's an odds-on bet that somebody on the project team (in this case, the performance engineer) will need it also. Again, the project benefits because it enters development better prepared.

Uncovers Opportunities for Cross-Project Reuse

Stakeholders and the evaluation team come from outside the development project, but often work on or are familiar with other projects within the same parent organization. As such, both are in a good position either to spot components that can be reused on other projects or to know of components (or other assets) that already exist and perhaps could be imported into the current project.

Results in Improved Architecture Practices

Organizations that practice architecture evaluation as a standard part of their development process report an improvement in the quality of the architectures that are evaluated. As development organizations learn to anticipate the kinds of questions that will be asked, the kinds of issues that will be raised, and the kinds of documentation that will be required for evaluations, they naturally preposition themselves to maximize their performance on the evaluations. Architecture evaluations result in better architectures not only after the fact but before the fact as well. Over time, an organization develops a culture that promotes good architectural design.

Now, not all of these benefits may resonate with you. If your organization is small, maybe all of the stakeholders know each other and talk regularly. Perhaps your organization is very mature when it comes to working out the requirements for a system, and by the time the finishing touches are put on the architecture the requirements are no longer an issue because everyone is completely clear what they are. If so, congratulations. But many of the organizations in which we have carried out architecture evaluations are not quite so sophisticated, and there have always been requirements issues that were raised (and resolved) when the architecture was put on the table.

There are also benefits to future projects in the same organization. A critical part of the ATAM consists of probing the architecture using a set of quality-specific analysis questions, and neither the method nor the list of questions is a secret. The architect is perfectly free to arm her- or himself before the evaluation by making sure that the architecture is up to snuff with respect to the relevant questions. This is rather like scoring well on a test whose questions you've already seen, but in this case it isn't cheating: it's professionalism.

The costs of architecture evaluation are all personnel costs and opportunity costs related to those personnel participating in the evaluation instead of something else. They're easy enough to calculate. An example using the cost of an ATAM-based evaluation is shown in Table 2.1. The left-most column names the phases of the ATAM (which will be described in subsequent chapters). The other columns split the cost among the participant groups. Similar tables can easily be constructed for other methods.

Table 2.1 shows figures for what we would consider a medium-size evaluation effort. While 70 person-days sounds like a substantial sum, in actuality it may not be so daunting. For one reason, the *calendar* time added to the project is minimal. The schedule should not be impacted by the preparation at all, nor the follow-up. These activities can be carried out behind the scenes, as it were. The middle phases consume actual project days, usually three or so. Second, the project normally does not have to pay for all 70 staff days. Many of the

Table 2.1 Approximate Cost of a Medium-Size ATAM-Based Evaluation

		Stakeholders	
Participant Group ATAM Phase	**Evaluation Team (assume 5 members)**	**Project Decision Makers (assume architect, project manager, customer)**	**Other Stakeholders (assume 8)**
Phase 0: Preparation	1 person-day by team leader	1 person-day	0
Phase 1: Initial evaluation (1 day)	5 person-days	3 person-days	0
Phase 2: Complete evaluation (3 days)	15 person-days	9 person-days + 2 person-days to prepare	16 person-days (most stakeholders present only for 2 days)
Phase 3: Follow-up	15 person-days	3 person-days to read and respond to report	0
TOTAL	36 person-days	18 person-days	16 person-days

stakeholders work for other cost centers, if not other organizations, than the development group. Stakeholders by definition have a vested interest in the system, and they are often more than willing to contribute their time to help produce a quality product.

It is certainly easy to imagine larger and smaller efforts than the one characterized by Table 2.1. As we will see, all of the methods are flexible, structured to iteratively spiral down into as much detail as the evaluators and evaluation client feel is warranted. Cursory evaluations can be done in a day; excruciatingly detailed evaluations could take weeks. However, the numbers in Table 2.2 represent what we would call nominal applications of the ATAM. For smaller projects, Table 2.2 shows how those numbers can be halved.

If your group evaluates many systems in the same domain or with the same architectural goals, then there is another way that the cost of evaluation can be reduced. Collect and record the scenarios used in each evaluation. Over time, you will find that the scenario sets will begin to resemble each other. After you have performed several of these almost-alike evaluations, you can produce a "canonical" set of scenarios based on past experience. At this point, the scenarios have in essence graduated to become a checklist, and you can dispense with the bulk of the scenario-generation part of the exercise. This saves about a day. Since scenario generation is the primary duty of the stakeholders, the bulk of their time can also be done away with, lowering the cost still further.

Table 2.2 Approximate Cost of a Small ATAM-Based evaluation

		Stakeholders	
Participant Group ATAM Phase	**Evaluation team (assume 2 members)**	**Project Decision Makers (assume architect, project manager)**	**Other Stakeholders (assume 3)**
Phase 0: Preparation	1 person-day by team leader	1 person-day	0
Phase 1: Initial evaluation (1 day)	2 person-days	2 person-days	0
Phase 2: Complete evaluation (2 days)	4 person-days	4 person-days + 2 person-days to prepare	6 person-days
Phase 3: Follow-up	8 person-days	2 person-days to read and respond to report	0
TOTAL	15 person-days	11 person-days	6 person-days

Table 2.3 Approximate Cost of a Medium-Size Checklist-based ATAM-Based Evaluation

		Stakeholders	
Participant Group ATAM Phase	**Evaluation Team (assume 4 members)**	**Project Decision Makers (assume architect, project manager, customer)**	**Other Stakeholders (assume the customer validates the checklist)**
Phase 0: Preparation	1 person-day by team leader	1 person-day	0
Phase 1: Initial evaluation (1 day)	4 person-days	3 person-days	0
Phase 2: Complete evaluation (2 days)	8 person-days	6 person-days	2 person-days
Phase 3: Follow-up	12 person-days	3 person-days to read and respond to report	0
TOTAL	25 person-days	13 person-days	2 person-days

(You still may want to have a few key stakeholders, including the customer, to validate the applicability of your checklist to the new system.) The team size can be reduced, since no one is needed to record scenarios. The architect's preparation time should be minimal since the checklist will be publicly available even when he or she begins the architecture task.

Table 2.3 shows the cost of a medium-size checklist-based evaluation using the ATAM, which comes in at about $^4/_7$ of the cost of the scenario-based evaluation of Table 2.1.

The next chapter will introduce the first of the three architecture evaluation methods in this book: the Architecture Tradeoff Analysis Method.

2.9 For Further Reading

The For Further Reading list of Chapter 9 (Comparing Software Architecture Evaluation Methods) lists good references on various architecture evaluation methods.

Zhao has assembled a nice collection of literature resources dealing with software architecture analysis [Zhao 99].

Once an architecture evaluation has identified changes that should be made to an architecture, how do you prioritize them? Work is emerging to help an architect or project manager assign quantitative cost and benefit information to architectural decisions [Kazman 01].

2.10 Discussion Questions

1. How does your organization currently decide whether a proposed software architecture should be adopted or not? How does it decide when a software architecture has outlived its usefulness and should be discarded in favor of another?
2. Make a business case, specific to your organization, that tells whether or not conducting a software architecture evaluation would pay off. Assume the cost estimates given in this chapter if you like, or use your own.
3. Do you know of a case where a flawed software architecture led to the failure or delay of a software system or project? Discuss what caused the problem and whether a software architecture evaluation might have prevented the calamity.
4. Which quality attributes tend to be the most important to systems in your organization? How are those attributes specified? How does the architect know what they are, what they mean, and what precise levels of each are required?
5. For each quality attribute discussed in this chapter—or for each that you named in answer to the previous question—hypothesize three different architectural decisions that would have an effect on that attribute. For example, the decision to maintain a backup database would probably increase a system's availability.
6. Choose three or four pairs of quality attributes. For each pair (think about tradeoffs), hypothesize an architectural decision that would increase the first quality attribute at the expense of the second. Now hypothesize a different architectural decision that would raise the second but lower the first.