UNIT-V

ANALYZING ARCHITECTURES

The ATAM :

In this chapter, we will introduce the Architecture Tradeoff Analysis Method (ATAM), a thorough and comprehensive way to evaluate a software architecture. The ATAM is so named because it reveals how well an architecture satisfies particular quality goals, and (because it recognizes that architectural decisions tend to affect more than one quality attribute) it provides insight into how quality goals interact?that is, how they trade off.

Evaluating an architecture for a large system is a complicated undertaking. First, a large system will have a comparably large architecture that will be difficult to understand in a limited amount of time. Second, according to Nietzsche and the Architecture Business Cycle (ABC), a computer system is intended to support business goals and the evaluation will need to make connections between those goals and the technical decisions. Finally, a large system usually has multiple stakeholders and acquiring their different perspectives in a limited amount of time requires careful management of an evaluation process. As you can see from this set of difficulties, managing the limited time for an architecture evaluation is a central problem.

Note: Mark Klein is a senior member of the technical staff at the Software Engineering Institute.

The ATAM is designed to elicit the business goals for the system as well as for the architecture. It is also designed to use those goals and stakeholder participation to focus the attention of the evaluators on the portion of the architecture that is central to the achievement of the goals.

This chapter will introduce the steps of the ATAM and discuss them in light of their intended purpose. It will also presents an ATAM case study (based on one of our applications of the method).

PARTICIPANTS IN THE ATAM :

The ATAM requires the participation and mutual cooperation of three groups:

1. The evaluation team. This group is external to the project whose architecture is being evaluated. It usually consists of three to five people. Each member of the team is assigned a number of specific roles to play during the evaluation. (See Table 5.1 for a description of these roles, along with a set of desirable characteristics for each.) The evaluation team may be a standing unit in which architecture evaluations are regularly performed, or its members may be chosen from a pool of architecturally savvy individuals for the occasion. They may work for the same organization as the development team whose architecture is on the table, or they may be outside consultants. In any case, they need to be recognized as competent, unbiased outsiders with no hidden agendas or axes to grind.
2. Project decision makers. These people are empowered to speak for the development project or have the authority to mandate changes to it. They usually include the project manager, and, if there is an identifiable customer who is footing the bill for the development, he or she will be present (or represented) as well. The architect is always included?a cardinal rule of architecture evaluation is that the architect must willingly participate. Finally, the person commissioning the evaluation is usually empowered to speak for the development project; even if not, he or she should be included in the group.
3. Architecture stakeholders. Stakeholders have a vested interest in the architecture performing as advertised. They are the ones whose ability to do their jobs hinges on the architecture promoting modifiability, security, high reliability, or the like. Stakeholders include developers, testers, integrators, maintainers, performance engineers, users, builders of systems interacting with the one under consideration, and others. Their job during an evaluation is to articulate the specific quality attribute goals that the architecture should meet in order for the system to be considered a success.

A rule of thumb?and that is all it is?is that you should expect to enlist the services of twelve to fifteen stakeholders for the evaluation.

Table 5.1. ATAM evaluation team roles

Role	**Responsibilities**	**Desirable characteristics**
Team Leader	Sets up the evaluation; coordinates with client, making sure client's needs are met; establishes evaluation contract; forms evaluation team; sees that final report is produced and delivered (although the writing may be delegated)	Well-organized, with managerial skills; good at interacting with client; able to meet deadlines
Evaluation Leader	Runs evaluation; facilitates elicitation of scenarios; administers scenario selection/prioritization process; facilitates evaluation of scenarios against architecture; facilitates onsite analysis	Comfortable in front of audience; excellent facilitation skills; good understanding of architectural issues; practiced in architecture evaluations; able to tell when protracted discussion is leading to a valuable discovery or when it is pointless and should be re-directed
Scenario Scribe	Writes scenarios on flipchart or whiteboard during scenario elicitation; captures agreed-on wording of each scenario, halting discussion until exact wording is captured	Good handwriting; stickler about not moving on before an idea (scenario) is captured; can absorb and distill the essence of technical discussions
Proceedings Scribe	Captures proceedings in electronic form on laptop or workstation, raw scenarios, issue(s) that motivate each scenario (often lost in the	Good, fast typist; well organized for rapid recall of information; good understanding of architectural

Table 5.1. ATAM evaluation team roles

Role	Responsibilities	Desirable characteristics
	wording of the scenario itself), and resolution of each scenario when applied to architecture(s); also generates a printed list of adopted scenarios for handout to all participants	issues; able to assimilate technical issues quickly; unafraid to interrupt the flow of discussion (at opportune times) to test understanding of an issue so that appropriate information is captured
Timekeeper	Helps evaluation leader stay on schedule; helps control amount of time devoted to each scenario during the evaluation phase	Willing to interrupt discussion to call time
Process Observer	Keeps notes on how evaluation process could be improved or deviated from; usually keeps silent but may make discreet process-based suggestions to the evaluation leader during the evaluation; after evaluation, reports on how the process went and lessons learned for future improvement; also responsible for reporting experience to architecture evaluation team at large	Thoughtful observer; knowledgeable in the evaluation process; should have previous experience in the architecture evaluation method
Process Enforcer	Helps evaluation leader remember and carry out the steps of the evaluation method	Fluent in the steps of the method, and willing and able to provide discreet guidance to the evaluation leader
Questioner	Raise issues of architectural interest that stakeholders may not have thought of	Good architectural insights; good insights into needs of stakeholders;

OUTPUTS OF THE ATAM

An ATAM-based evaluation will produce at least the following outputs:

- A concise presentation of the architecture. Architecture documentation is often thought to consist of the object model, a list of interfaces and their signatures, or some other voluminous list. But one of the requirements of the ATAM is that the architecture be presented in one hour, which leads to an architectural presentation that is both concise and, usually, understandable.
- Articulation of the business goals. Frequently, the business goals presented in the ATAM are being seen by some of the development team for the first time.
- Quality requirements in terms of a collection of scenarios. Business goals lead to quality requirements. Some of the important quality requirements are captured in the form of scenarios.
- Mapping of architectural decisions to quality requirements. Architectural decisions can be interpreted in terms of the qualities that they support or hinder. For each quality scenario examined during an ATAM, those architectural decisions that help to achieve it are determined.
- A set of identified sensitivity and tradeoff points. These are architectural decisions that have a marked effect on one or more quality attributes. Adopting a backup database, for example, is clearly an architectural decision as it affects reliability (positively), and so it is a sensitivity point with respect to reliability.

  However, keeping the backup current consumes system resources and so affects performance negatively. Hence, it is a tradeoff point between reliability and performance. Whether this decision is a risk or a nonrisk depends on whether its performance cost is excessive in the context of the quality attribute requirements of the architecture.

- A set of risks and nonrisks. A risk is defined in the ATAM as an architectural decision that may lead to undesirable consequences in light of stated quality attribute requirements. Similarly, a nonrisk is an architectural decision that, upon analysis, is deemed safe. The identified risks can form the basis for an architectural risk mitigation plan.
- A set of risk themes. When the analysis is complete, the evaluation team will examine the full set of discovered risks to look for over-arching themes that identify systemic weaknesses in the architecture or even in the architecture process and team. If left untreated, these risk themes will threaten the project's business goals.

The outputs are used to build a final written report that recaps the method, summarizes the proceedings, captures the scenarios and their analysis, and catalogs the findings.

There are secondary outputs as well. Very often, representations of the architecture will have been created expressly for the evaluation and may be superior to whatever existed before. This additional documentation survives the evaluation and can become part of the project's legacy. Also, the scenarios created by the participants are expressions of the business goals and requirements for the architecture and can be used to guide the architecture's evolution. Finally, the analysis contained in the final report can serve as a statement of rationale for certain architectural decisions made (or not made). The secondary outputs are tangible and enumerable.

There are intangible results of an ATAM-based evaluation. These include a palpable sense of community on the part of the stakeholders, open communication channels between the architect and the stakeholders, and a better overall understanding on the part of all participants of the architecture and its strengths and weaknesses. While these results are hard to measure, they are no less important than the others and often are the longest-lasting.

PHASES OF THE ATAM :

Activities in an ATAM-based evaluation are spread out over four phases.

In phase 0, "Partnership and Preparation," the evaluation team leadership and the key project decision makers informally meet to work out the details of the exercise.

The project representatives brief the evaluators about the project so that the team can be supplemented by people who possess the appropriate expertise. Together, the two groups agree on logistics, such as the time and place of meetings, who brings the flipcharts, and who supplies the donuts and coffee. They also agree on a preliminary list of stakeholders (by name, not just role), and they negotiate on when the final report is to be delivered and to whom. They handle formalities such as a statement of work or nondisclosure agreements. They work out delivery to the evaluation team of whatever architectural documentation exists and may be useful. Finally, the evaluation team leader explains what information the manager and architect will be expected to show during phase 1, and helps them construct their presentations if necessary.

Phase 1 and phase 2 are the evaluation phases, where everyone gets down to the business of analysis. By now the evaluation team will have studied the architecture documentation and will have a good idea of what the system is about, the overall architectural approaches taken, and the quality attributes that are of paramount importance. During phase 1, the evaluation team meets with the project decision makers (usually for about a day) to begin information gathering and analysis. For phase 2, the architecture's stakeholders join the proceedings and analysis continues, typically for two days. The exact steps of phase 1 and phase 2 are detailed in the next section.

Phase 3 is follow-up in which the evaluation team produces and delivers a written final report. The essence of this phase, however, is team self-examination and improvement. During a post-mortem meeting, the team discusses what went well and what didn't. They study the surveys handed out to participants during phase 1 and phase 2, and the process observer makes his or her report. Team members look for improvements in how they carry out their functions so that the next evaluation can be smoother or more effective. The team catalogs how much effort was spent during the evaluation, on the part of each of the three participating groups. After an appropriate number of months, the team leader contacts the evaluation client to gauge the long-term effects of the exercise so that costs and benefits can be compared.

Table 5.2 shows the four phases of the ATAM, who participates in each one, and an approximate timetable.

Table 5.2. ATAM Phases and Their Characteristics

Phase	**Activity**	**Participants**	**Typical Duration**
0	Partnership and preparation	Evaluation team leadership and key project decision makers	Proceeds informally as required, perhaps over a few weeks
1	Evaluation	Evaluation team and project decision makers	1 day followed by a hiatus of 2 to 3 weeks
2	Evaluation (continued)	Evaluation team, project decision makers, and stakeholders	2 days
3	Follow-up	Evaluation team and evaluation client	1 week

THE CBAM:

The Architecture Tradeoff Analysis Method (ATAM) provides software architects a means of evaluating the technical tradeoffs faced while designing or maintaining a software system. In the ATAM, we are primarily investigating how well the architecture?real or proposed?has been designed with respect to the quality attributes that its stakeholders have deemed important. We are also analyzing architectural tradeoffs?the places where a decision might have consequences for several quality attributes simultaneously.

However, the ATAM is missing an important consideration: The biggest tradeoffs in large, complex systems usually have to do with economics. How should an organization invest its resources in a manner that will maximize its gains and minimize its risk? In the past, this question primarily focused on costs, and even then these were primarily the costs of building the system in the first place and not the long-term costs through cycles of maintenance and upgrade. As important, or perhaps more important than costs, are the benefits that an architectural decision may bring to an organization.

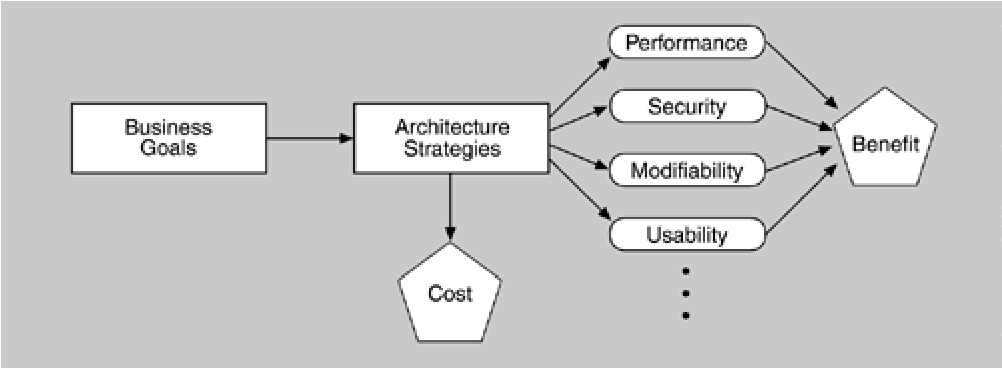
Given that the resources for building and maintaining a system are finite, there must be a rational process that helps us choose among architectural options, during both an initial design phase and subsequent upgrade periods. These options will have different costs, will consume differing amounts of resources, will implement different features (each of which brings some benefit to the organization), and will have some inherent risk or uncertainty. To capture these aspects we need economic models of software that take into account costs, benefits, risks, and schedule implications.

To address this need for economic decision making, we have developed a method of economic modeling of software systems, centered on an analysis of their architectures. Called the Cost Benefit Analysis Method (CBAM), it builds on the ATAM to model the costs and the benefits of architectural design decisions and is a means of optimizing such decisions. The CBAM provides an assessment of the technical and economic issues and architectural decisions.

DECISION MAKING CONTEXT :

The software architect or decision maker wishes to maximize the difference between the benefit derived from the system and the cost of implementing the design. The CBAM begins where the ATAM concludes and, in fact, depends upon the artifacts that the ATAM produces as output. Figure 5.1 depicts the context for the CBAM.

Figure 5.1. Context for the CBAM



Because architectural strategies have technical and economic implications, the business goals of a software system should influence the strategies used by software architects or designers.

The direct economic implication is the cost of implementing the system. The technical implications are the characteristics of the system?namely, the quality attributes. In turn the quality attributes have economic implications because of the benefits that can be derived.

Recall that when an ATAM has been applied to a software system, we have as a result a set of artifacts documented on completion. They are:

- A description of the business goals that are crucial to the success of the system
- A set of architectural views that document the existing or proposed architecture
- A utility tree that represents a decomposition of the stakeholders' goals for the architecture, starting with high-level statements of quality attributes and ending with specific scenarios
- A set of risks that have been identified
- A set of sensitivity points (architectural decisions that affect some quality attribute measure of concern)
- A set of tradeoff points (architectural decisions that affect more than one quality attribute measure, some positively and some negatively)

The ATAM identifies the set of key architectural decisions relevant to the quality attribute scenarios elicited from the stakeholders. These decisions result in some specific quality attribute responses?namely, particular levels of availability, performance, security, usability, modifiability, and so forth. But each architectural decision also has associated costs. For example, using redundant hardware to achieve a desired level of availability has a cost; checkpointing to a disk file has a different cost. Furthermore, both of these architectural decisions will result in (presumably different) measurable levels of availability that will have some value to the organization developing the system. Perhaps the organization believes that its stakeholders will pay more for a highly available system (a telephone switch or medical monitoring software, for example) or that it will be sued if the system fails (for example, the software that controls anti-lock brakes in an automobile).

The ATAM uncovers the architectural decisions made in the system and links them to business goals and quality attribute response measures. The CBAM builds on this base by eliciting the costs and benefits associated with these decisions. Given this information, the stakeholders can then decide whether to use redundant hardware, checkpointing, or some other tactic to achieve the system's desired availability. Or they can choose to invest their finite resources in some other quality attribute?perhaps believing that higher performance will have a better benefit-to-cost ratio. A system always has a limited budget for creation or upgrade, so every architectural choice is, in some sense, competing with every other one for inclusion.

The CBAM does not make decisions for the stakeholders, just as a financial advisor does not tell you how to invest your money. It simply aids in the elicitation and documentation of the costs, benefits, and uncertainty of a "portfolio" of architectural investments and gives the stakeholders a framework within which they can apply a rational decision-making process that suits their needs and their risk aversion.

To briefly summarize, the idea behind the CBAM is that architectural strategies (a collection of architectural tactics) affect the quality attributes of the system and these in turn provide system stakeholders with some benefit. We refer to this benefit as utility. Each architectural strategy provides a specific level of utility to the stakeholders.

Each also has cost and takes time to implement. Given this information, the CBAM can aid the stakeholders in choosing architectural strategies based on their return on investment (ROI)?the ratio of benefit to cost.

The Basis for the CBAM

We now describe the key ideas that form the basis for the CBAM. The practical realization of these ideas as a series of steps will be described in Section 5.3. Our goal here is to develop the theory underpinning a measure of ROI for various architectural strategies in light of scenarios chosen by the stakeholders.

We begin by considering a collection of scenarios generated either as a portion of an ATAM or especially for the CBAM evaluation.

We examine how they differ in the values of their projected responses and then assign utility to those values. The utility is based on the importance of each scenario being considered with respect to its anticipated response value. We next consider the architectural strategies that lead to the various projected responses. Each strategy has a cost, and each impacts multiple quality attributes. That is, an architectural strategy could be implemented to achieve some projected response, but while achieving that response it also affects some other quality attributes. The utility of these "side effects" must be taken into account when considering a strategy's overall utility. It is this overall utility that we combine with the project cost of an architectural strategy to calculate a final ROI measure.

UTILITY

Utility is determined by considering the issues described in the following sections.

Variations of Scenarios

The CBAM uses scenarios as a way to concretely express and represent specific quality attributes, just as in the ATAM. Also as in the ATAM, we structure scenarios into three parts: stimulus (an interaction with the system), environment (the system's state at the time), and response (the measurable quality attribute that results).

However, there is a difference between the methods:

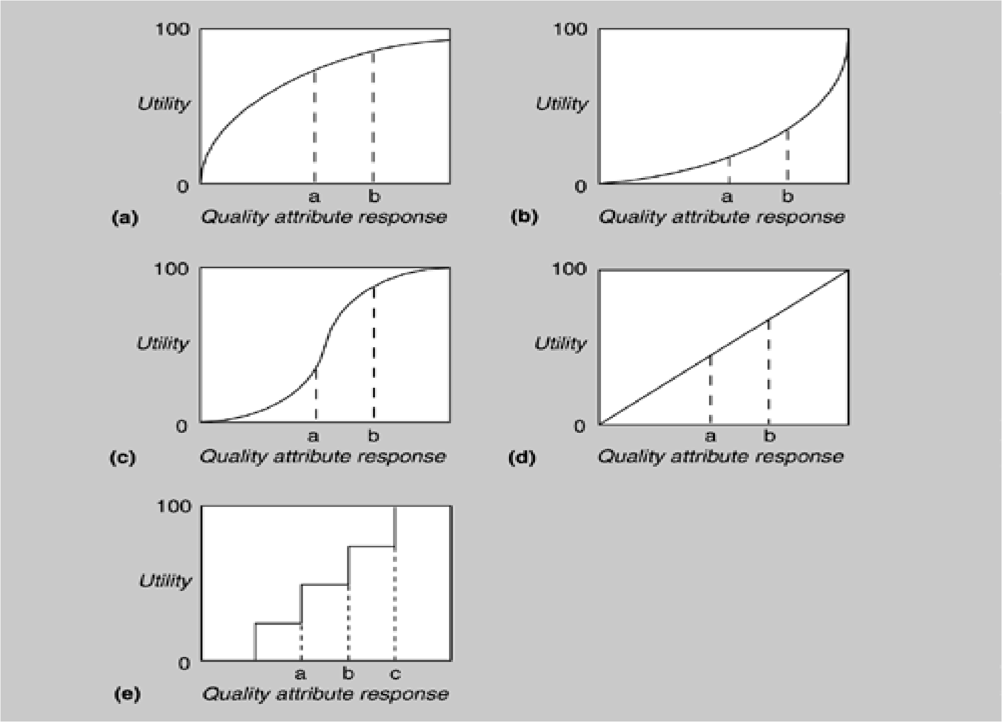
The CBAM actually uses a set of scenarios (generated by varying the values of the responses) rather than individual scenarios as in the ATAM. This leads to the concept of a utility-response curve.

Utility-Response Curves

Every stimulus-response value pair in a scenario provides some utility to the stakeholders, and the utility of different possible values for the response can be compared. For example, a very high availability in response to failure might be valued by the stakeholders only slightly more than moderate availability. But low latency might be valued substantially more than moderate latency.

We can portray each relationship between a set of utility measures and a corresponding set of response measures as a graph?a utility-response curve. Some examples of utility-response curves are shown in Figure 5.2. In each, points labeled a, b, or c represent different response values. The utility-response curve thus shows utility as a function of the response value.

Figure 5.2. Some sample utility-response curves



The utility-response curve depicts how the utility derived from a particular response varies as the response varies. As seen in Figure 5.2, the utility could vary nonlinearly, linearly, or even as a step-function. For example, graph (c) portrays a steep rise in utility over a narrow change in a quality attribute response level, such as the performance example stated above. The availability example might be better characterized by graph (a), where a modest change in the response level results in only a very small change in utility to the user.

Eliciting the utility characteristics from the stakeholders can be a long and tedious process. To make it practical we have chosen to elicit only rough approximations of these curves from the stakeholders, using five values of the quality attribute response for the scenario.

We now explain the four of these values that can be derived without consideration of any architectural strategy. The fifth value depends on the architectural strategy used, and we discuss this later.

To build the utility-response curve, we first determine the quality attribute levels for the best-case and worst-case situations. The best-case quality attribute level is that above which the stakeholders foresee no further utility. For example, a system response to the user of 0.1 second is perceived as instantaneous, so improving it further so that it responds in 0.03 second has no utility. Similarly, the worst-case quality attribute level is a minimum threshold above which a system must perform; otherwise it is of no use to the stakeholders. These levels?best-case and worst-case?are assigned utility values of 100 and 0, respectively.

We must then determine the current and desired utility levels for the scenario. The respective utility values (between 0 and 100) for the current and desired cases are elicited from the stakeholders, using the best-case and worst-case values as reference points (e.g., we are currently half as good as we would like to be, but if we reach the desired quality attribute level, we will have 90% of the maximum utility; hence, the current utility level is set to 50 and the desired utility level is set to 90). In this manner the curves are generated for all of the scenarios.

Priorities of Scenarios

Different scenarios within a given system have different levels of importance to the stakeholders and hence different utilities.

To characterize the relative importance of each scenario, a weight is assigned through a two-step voting exercise. In the first step the stakeholders vote on the scenarios to establish an ordering among them. This voting is based on each scenario's "expected" response value. The stakeholders then assign a weight of 1 to the highest-rated scenario and a fractional amount to the other scenarios based on their relative importance.

If, at some future date, additional scenarios need to be added, they can be assigned a weight. The stakeholders, through consensus, can make sure that the scenarios weights accord with their intuition.

Architectural Strategies

It is the job of the architect, or architects, to determine the architectural strategies for moving from the current quality attribute response level to the desired or even best-case level. A portion of the CBAM is devoted to this task. For each strategy, we can derive

- the expected value of the response in each scenario. The utility of the expected value is calculated using interpolation from the four values already elicited from the stakeholders.
- the effect of the architectural strategy on other attributes of interest.
- a cost estimate for implementing the architectural strategy.

Side effects

Each architectural strategy will impact not only the quality attribute from the scenario being considered currently but will typically also affect other quality attributes (this is why there are architectural tradeoffs!). It is important to determine the utility of these additional side effect attribute responses that arise as a result of applying the architectural strategy. In the worst case, we must create a new version of the scenario for the side effect attribute and determine its utility-response curve. However, in practice, if the quality attribute is important to the stakeholders, then it has occurred in one of the other scenarios and the utility-response curve has already been constructed for that response.

In this case, the only thing left to determine is the expected utility associated with that quality attribute for the given architectural strategy. Notice that it is possible that the expected utility for a particular attribute may be negative if the architectural strategy is designed to emphasize an attribute in conflict with the one whose utility we are currently calculating.

Once this additional information has been elicited we can calculate the benefit of applying an architectural strategy by summing its benefits to all relevant quality attributes.

Determining benefit and normalization

We calculate the overall utility of an architectural strategy across scenarios from the utility-response curves by summing the utility associated with each one (weighted by the importance of the scenario). For each architectural strategy, i, we calculate a benefit, B_i as follows:

$$B_i = \sum_j (b_{i,j} \times W_j)$$

where $b_{i,j}$ is the benefit accrued to strategy i due to its effect on scenario j and W_j is the weight of scenario j. Referring to Figure 12.2, each $b_{i,j}$ is calculated as the change in utility brought about by the architectural strategy with respect to this scenario: $b_{i,j} = U_{expected}$? $U_{current}$; that is, the utility of the expected value of the architectural strategy minus the utility of the current system relative to this scenario. The effect of multiplying the weight, W_j, is to normalize this utility value by the relative importance of the various scenarios, as already described.

CALCULATING ROI

The ROI value for each architectural strategy is the ratio of the total benefit, B_i, to the Cost, C_i, of implementing it. The cost is calculated using a model appropriate for the system and the environment being developed.

$$R_i = \frac{B_i}{C_i}$$

Using this ROI score, the architectural strategies can be rank-ordered; this rank ordering can then be used to determine the optimal order for implementation of the various strategies.

Consider curves (a) and (b) in Figure 12.2. Curve (a) "flattens out" as the quality attribute response improves. In this case, it is likely that a point is reached past which ROI decreases as the quality attribute response improves. In other words, spending more money will not yield a significant increase in utility. On the other hand, consider curve (b), for which a small improvement in quality attribute response can yield a very significant increase in utility. There an architectural strategy whose ROI is too low might rank significantly higher with a modest improvement in its quality attribute response.

Implementing the CBAM

Turning the foundations for the CBAM into a set of practical steps involves taking the bases we discussed in the previous section and performing them in a fashion that minimizes the work that is needed. Part of being "practical" involves limiting the size of the decision space.

STEPS :

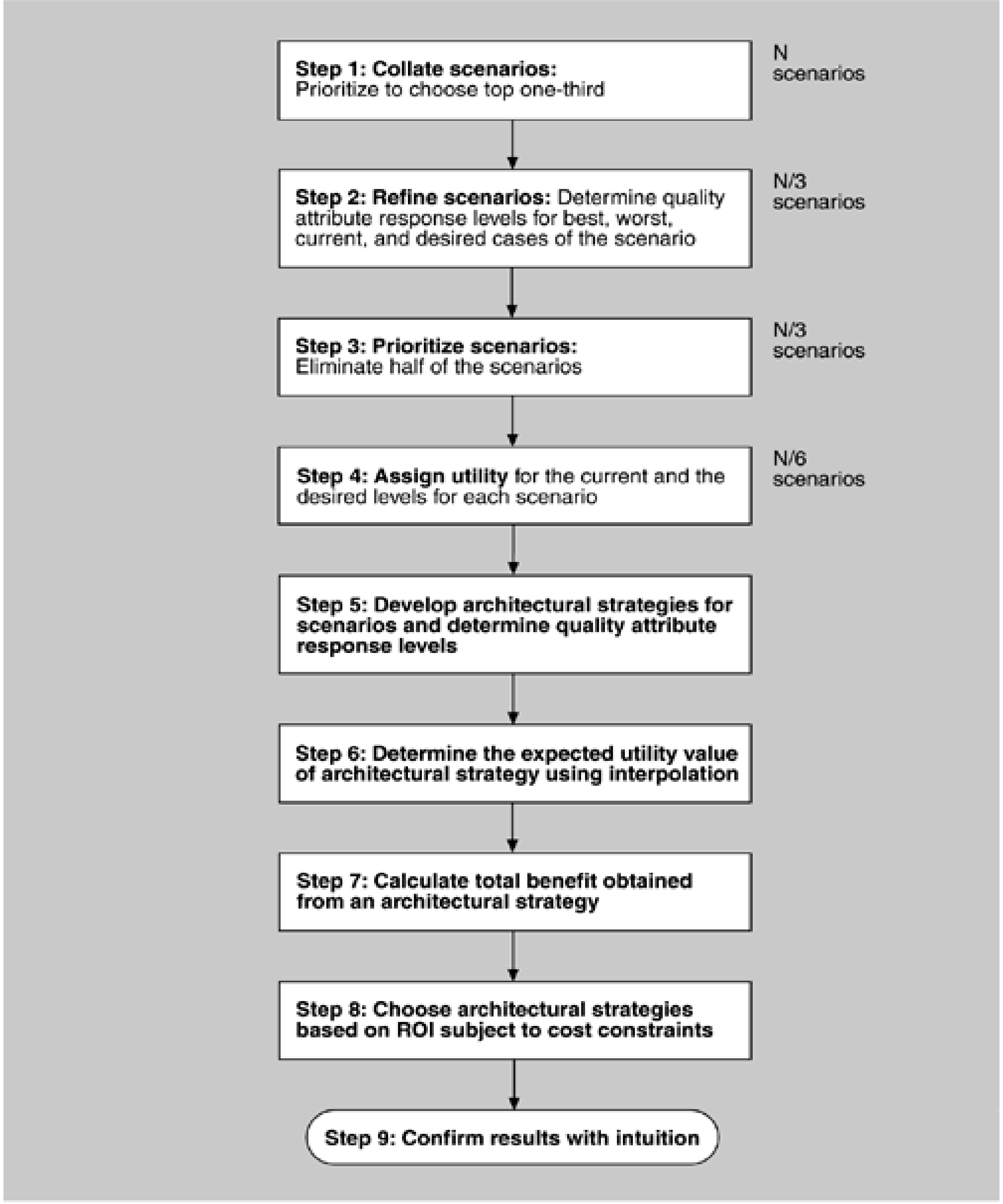
A process flow diagram for the CBAM is given in Figure 5.3. The first four steps are annotated with the relative number of scenarios they consider. That number steadily decreases, ensuring that the method concentrates the stakeholders' time on the scenarios believed to be of the greatest potential in terms of ROI.

- Step 1: Collate scenarios. Collate the scenarios elicited during the ATAM exercise, and give the stakeholders the chance to contribute new ones. Prioritize these scenarios based on satisfying the business goals of the system and choose the top one-third for further study.
- Step 2: Refine scenarios. Refine the scenarios output from step 1, focusing on their stimulus-response measures. Elicit the worst-case, current, desired, and best-case quality attribute response level for each scenario.
- Step 3: Prioritize scenarios. Allocate 100 votes to each stakeholder and have them distribute the votes among the scenarios, where their voting is based on the desired response value for each scenario. Total the votes and choose the top 50% of the scenarios for further analysis. Assign a weight of 1.0 to the highest-rated scenario; assign the other scenarios a weight relative to the highest rated. This becomes the weighting used in the calculation of a strategy's overall benefit. Make a list of the quality attributes that concern the stakeholders.
- Step 4: Assign utility. Determine the utility for each quality attribute response level (worst-case, current, desired, best-case) for the scenarios from step 3.

- Step 5: Develop architectural strategies for scenarios and determine their expected quality attribute response levels. Develop (or capture already developed) architectural strategies that address the chosen scenarios and determine the "expected" quality attribute response levels that will result from them. Given that an architectural strategy may have effects on multiple scenarios, we must perform this calculation for each scenario affected.
- Step 6: Determine the utility of the "expected" quality attribute response levels by interpolation. Using the elicited utility values (that form a utility curve), determine the utility of the expected quality attribute response level for the architectural strategy. Do this for each relevant quality attribute enumerated in step 3.
- Step 7: Calculate the total benefit obtained from an architectural strategy. Subtract the utility value of the "current" level from the expected level and normalize it using the votes elicited in step 3. Sum the benefit due to a particular architectural strategy across all scenarios and across all relevant quality attributes.
- Step 8: Choose architectural strategies based on ROI subject to cost and schedule constraints. Determine the cost and schedule implications of each architectural strategy. Calculate the ROI value for each as a ratio of benefit to cost.

  Rank-order the architectural strategies according to the ROI value and choose the top ones until the budget or schedule is exhausted.
- Step 9: Confirm results with intuition. For the chosen architectural strategies, consider whether these seem to align with the organization's business goals. If not, consider issues that may have been overlooked while doing this analysis. If there are significant issues, perform another iteration of these steps.

Figure 5.3. Process flow diagram for the CBAM



The World Wide Web'A Case Study in Interoperability

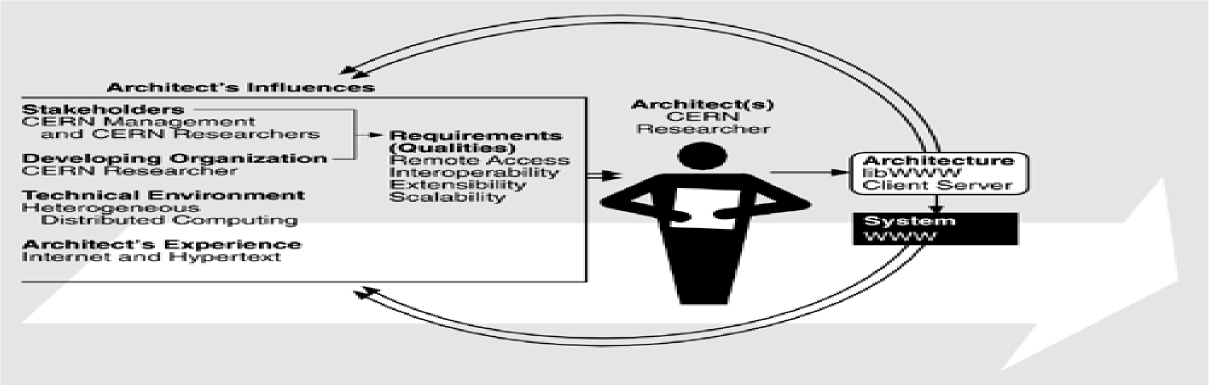
Relationship to the Architecture Business Cycle

The original proposal for the Web came from Tim Berners-Lee, a researcher with the European Laboratory for Particle Physics (CERN), who observed that the several thousand researchers at CERN formed an evolving human "web."

People came and went, developed new research associations, lost old ones, shared papers, chatted in the hallways, and so on, and Berners-Lee wanted to support this informal web with a similar web of electronic information. In 1989, he created and circulated throughout CERN a document entitled Information Management: A Proposal. By October of 1990 a reformulated version of the project proposal was approved by management, the name World Wide Web was chosen, and development began.

Figure 5.2 shows the elements of the ABC as they applied to the initial proposal approved by CERN management. The system was intended to promote interaction among CERN researchers (the end users) within the constraints of a heterogeneous computing environment. The customer was CERN management, and the developing organization was a lone CERN researcher. The business case made by Berners-Lee was that the proposed system would increase communication among CERN staff. This was a very limited proposal with very limited (and speculative) objectives. There was no way of knowing whether such a system would, in fact, increase communication. On the other hand, the investment required by CERN to generate and test the system was also very limited: one researcher's time for a few months.

Figure 5.2. The original ABC for the Web



The technical environment was familiar to those in the research community, for which the Internet had been a mainstay since its introduction in the early 1970s. The net had weak notions of central control (volunteer committees whose responsibilities were to set protocols for communication among different nodes on the Internet and to charter new newsgroups) and an unregulated, "wild-west" style of interaction, primarily through specialized newsgroups.

Hypertext systems had had an even longer history, beginning with the vision of Vannevar Bush in the 1940s. Bush's vision had been explored throughout the 1960s and 1970s and into the 1980s, with hypertext conferences held regularly to bring researchers together. However, Bush's vision had not been achieved on a large scale by the 1980s: The uses of hypertext were primarily limited to small-scale documentation systems. That was to change.

CERN management approved Berners-Lee's proposal in October 1990. By November he had developed the first Web program on the NeXT platform, which meant he clearly had begun working on the implementation before receiving formal management approval. This loose coupling between management approval and researcher activity is quite common in research organizations in which small initial investments are required. By their nature, research organizations tend to generate projects from the bottom up more often than commercial organizations do, because they are dependent on the researchers' originality and creativity and allow far more freedom than is typical in a commercial organization.

The initial implementation of a Web system had many features that are still missing from more recent Web browsers. For example, it allowed users to create links from within the browser, and it allowed authors and readers to annotate information. Berners-Lee initially thought that no user would want to write HyperText Markup Language (HTML) or deal with uniform resource locators (URLs). He was wrong. Users have been willing to put up with these inconveniences to have the power of publishing on the Web.

Requirements and Qualities

The World Wide Web, as conceived and initially implemented at CERN, had several desirable qualities.

It was portable, able to interoperate with other types of computers running the same software, and was scalable and extensible. The business goals of promoting interaction and allowing heterogeneous computing led to the quality goals of remote access, interoperability, extensibility, and scalability, which in turn led to libWWW, the original software library that supported Web-based development and a distributed client-server architecture.

The realization of these properties in the original software architecture created an infrastructure that effectively supported the Web's tremendous growth (see Table 5.2). libWWW embodies strict separation of concerns and therefore works on virtually any hardware and readily accepts new protocols, new data formats, and new applications. Because it has no centralized control, the Web appears to be able to grow without bounds.

Table 5.2. Web Growth Statistics

Date	Number of Web Sites	Percentage of .com Sites	Hosts per Web Server
6/93	130	1.5	13,000
12/93	623	4.6	3,475
6/94	2,738	13.5	1,095
12/94	10,022	18.3	451
6/95	23,500	31.3	270
1/96	100,000	50.0	94
6/96	252,000	68.0	41
1/97	646,162	62.6	40
1/98	1,834,710		16.2
1/99	4,062,280		10.6
1/00	9,950,491		7.3
1/01	27,585,719	54.68	4.0

Source: Used with permission of Matthew Gray of the Massachusetts Institute of Technology.

We will deal with these core requirements, and others, in more detail now, returning to the structure of libWWW later in Section 13.3.

There is no explicit requirement for ease of use in the original requirements, and it was not until the development of point-and-click browsers that the Web began its tremendous growth. On the other hand, the requirement for portability and the heterogeneous computing environment led to the introduction of the browser as a separate element, thereby fostering the development of more sophisticated browsers.

THE ORIGINAL REQUIREMENTS

The initial set of requirements for the Web, as established in the original project proposals, were as follows:

- Remote access across networks. Any information had to be accessible from any machine on a CERN network.
- Heterogeneity. The system could not be limited to run on any specific hardware or software platform.
- Noncentralization. In the spirit of a human web and of the Internet, there could not be any single source of data or services. This requirement was in anticipation that the Web would grow. The operation of linking to a document, in particular, had to be decentralized.
- Access to existing data. Existing databases had to be accessible.
- Ability for users to add data. Users should be able to "publish" their own data on the Web, using the same interface used to read others' data.
- Private links. Links and nodes had to be capable of being privately annotated.
- Bells and whistles. The only form of data display originally planned was display on a 24 x 80 character ASCII terminal. Graphics were considered optional.
- Data analysis. Users should be able to search across the various databases and look for anomalies, regularities, irregularities, and so on. Berners-Lee gave, as examples, the ability to look for undocumented software and organizations with no people.

- Live links. Given that information changes all the time, there should be some way of updating a user's view of it. This could be by simply retrieving the information every time the link is accessed or (in a more sophisticated fashion) by notifying a user of a link whenever the information has changed.

In addition to these requirements, there were a number of non-requirements identified. For example, copyright enforcement and data security were explicitly mentioned as requirements that the original project would not deal with. The Web, as initially conceived, was to be a public medium. Also, the original proposal explicitly noted that users should not have to use any particular markup format.

Other criteria and features that were common in proposals for hypertext systems at the time but that were missing from the Web proposal are as follows:

- Controlling topology
- Defining navigational techniques and user interface requirements, including keeping a visual history
- Having different types of links to express differing relationships among nodes

Although many of the original requirements formed the essence of what the Web is today, several were not realized, were only partially realized, or their impact was dramatically underestimated. For example, data analysis, live links, and private link capabilities are still relatively crude to this day. These requirements have gone largely unfulfilled.

Adaptation and selective postponement of requirements are characteristic of unprecedented systems.

Requirements are often lists of desirable characteristics, and in unprecedented systems the tradeoffs required to realize these requirements are often unknown until a design exists. In the process of making the tradeoffs, some requirements become more important and others less so.

The effect of one of the requirements turned out to have been greatly underestimated. Namely, the "bells and whistles" of graphics dominate much of today's Web traffic. Graphics today carry the bulk of the interest and consume the bulk of the Internet traffic generated by the Web. And yet Berners-Lee and CERN management did not concern themselves with graphics in the initial proposal, and the initial Web browser was line oriented. Similarly, the original proposal eschewed any interest in multimedia research for supporting sound and video.

Some nonrequirements, as the ABC has been traversed, have also become requirements. Security, for one, has proven to be a substantial issue, particularly as the Web has become increasingly dominated by commercial traffic. The security issue is large and complex, given the distributed, decentralized form of the Internet. Security is difficult to ensure when protected access to private data cannot be guaranteed?the Web opens a window onto your computer, and some uninvited guests are sure to crawl through.

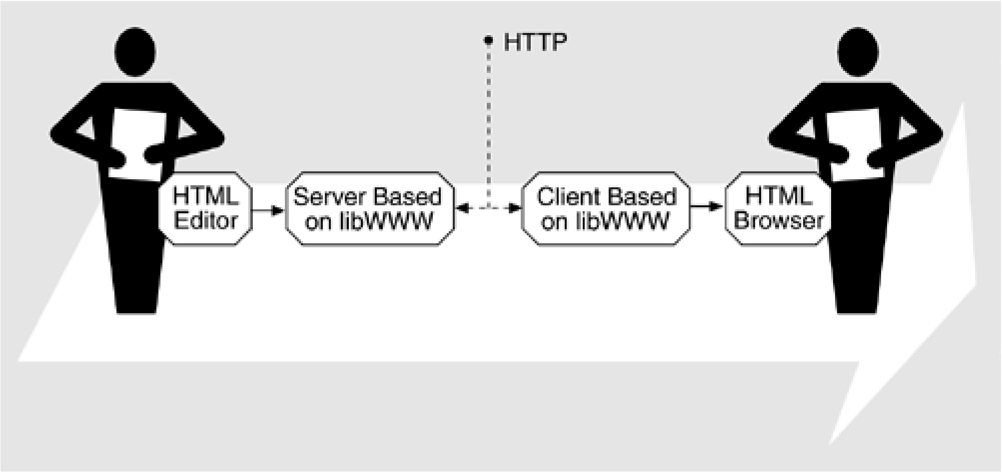
This has become even more relevant in recent years as e-commerce has begun to drive the structure and direction of the Web and a large number of ad hoc mechanisms have been created to facilitate it. The most obvious is simple encryption of sensitive data, typically via SSL (Secure Sockets Layer), seen in Web browsers as HTTPS (HyperText Transfer Protocol Secure). But this protocol only decreases the likelihood of others snooping on your private data while it is being transmitted over a public network. Other solutions?such as Microsoft's Passport?have you prove that you are who you say you are. (Chapter 4 discussed the various aspects of security, and Chapter 5 presented a set of tactics to achieve it.)

Architectural Solution

The basic architectural approach used for the Web, first at CERN and later at the World Wide Web Consortium (W3C), relied on clients and servers and a library (libWWW) that masks all hardware, operating system, and protocol dependencies. Figure 13.3 shows how the content producers and consumers interact through their respective servers and clients. The producer places content that is described in HTML on a server machine.

The server communicates with a client using the HyperText Transfer Protocol (HTTP). The software on both the server and the client is based on libWWW, so the details of the protocol and the dependencies on the platforms are masked from it. One of the elements on the client side is a browser that knows how to display HTML so that the content consumer is presented with an understandable image.

Figure 5.3. Content producers and consumers interact through clients and servers



We now go into more detail about both the libWWW and the client-server architecture used as the basis for the original Web and that still largely pervades Web-based software. Section 5.4 will discuss how the architecture of the Web and Web-based software have changed in response to the e-commerce revolution.

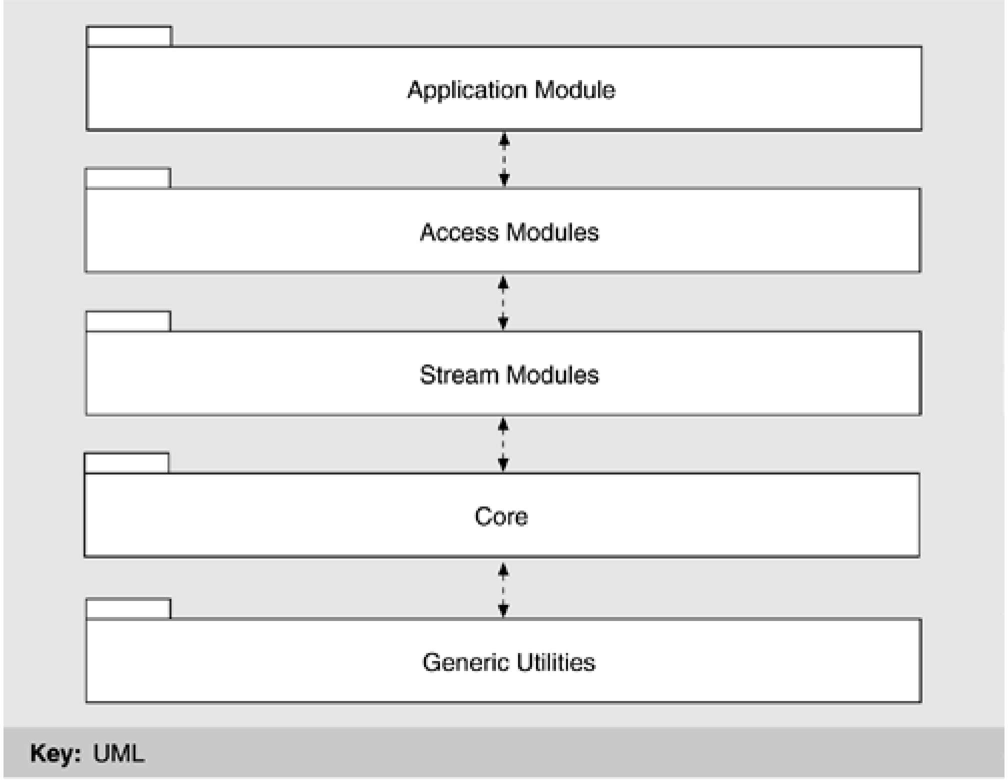
MEETING THE ORIGINAL REQUIREMENTS: libWWW

As stated earlier, libWWW is a library of software for creating applications that run on either the client or the server.

It provides the generic functionality that is shared by most applications: the ability to connect with remote hosts, the ability to understand streams of HTML data, and so forth.

libWWW is a compact, portable library that can be built on to create Web-based applications such as clients, servers, databases, and Web spiders. It is organized into five layers, as shown in Figure 5.4.

Figure 5.4. A layered view of libWWW



The generic utilities provide a portability layer on which the rest of the system rests. This layer includes basic building blocks for the system such as network management, data types such as container classes, and string manipulation utilities. Through the services provided by this layer, all higher levels can be made platform independent, and the task of porting to a new hardware or software platform can be almost entirely contained within the porting of the utilities layer, which needs to be done only once per platform.

The core layer contains the skeletal functionality of a Web application?network access, data management and parsing, logging, and the like. By itself, this layer does nothing.

Rather, it provides a standard interface for a Web application to be built upon, with the actual functionality provided by plug-in modules and call-out functions that are registered by an application. Plug-ins are registered at runtime and do the actual work of the core layer?sending and manipulating data.

They typically support protocols, handle low-level transport, and understand data formats. Plug-ins can be changed dynamically, making it easy to add new functionality or even to change the very nature of the Web application.

Achieving Quality Goals

Together the elements we have described allow the Web-based e-commerce system to achieve its stringent quality goals of security, high availability, modifiability, scalability, and high performance. How they do this is shown in Table 5.3.

Table 5.3. How the Web e-Commerce Architecture Achieves Its Quality Goals

Goal	**How Achieved**	**Tactics**
High Performance	Load balancing, network address translation, proxy servers	Introduce concurrency; increase resources; multiple copies
High Availability	Redundant processors, networks, databases, and software; load balancing	Active redundancy; transactions; introduce concurrency
Scalability	Allow for horizontal and vertical scaling; load balancing	Abstract common services; adherence to defined protocols; introduce concurrency
Security	Firewalls; public/private key encryption across public networks	Limit access; integrity; limit exposure
Modifiability	Separation of browser functionality, database design, and business logic into distinct tiers	Abstract common services; semantic coherence; intermediary; interface stability