Experience with Performing Architecture Tradeoff Analysis

Rick Kazman, Mario Barbacci, Mark Klein, S. Jeromy Carrière
Software Engineering Institute, Carnegie Mellon University
Pittsburgh, PA, U.S.A. 15213

+1-412-268-1588
{kazman, mrb, mk, sjc}@sei.cmu.edu

ABSTRACT
Software architectures, like complex designs in any field, embody tradeoffs made by the designers. However, these tradeoffs are not always made explicitly by the designers and they may not understand the impacts of their decisions. This paper describes the use of a scenario-based and model-based analysis technique for software architectures—called ATAM—that not only analyzes a software architecture with respect to multiple quality attributes, but explicitly considers the tradeoffs inherent in the design. This is a method aimed at illuminating risks in the architecture through the identification of attribute trends, rather than at precise characterizations of measurable quality attribute values. The ATAM is illustrated in this paper via an example where we analyzed a U.S. Army system for battlefield management.

Keywords
Architecture analysis, quality attribute models, architectural styles

1 WHY ARCHITECTURE TRADEOFF ANALYSIS?

At the Software Engineering Institute (SEI), we have been performing architectural analyses for the past 5 years, initially using the SAAM (Software Architecture Analysis Method) [6] and, more recently, using the ATAM (Architecture Tradeoff Analysis Method) [7]. The ATAM, like the SAAM, is a scenario-based method. However, unlike the SAAM, the ATAM focuses on multiple quality attributes (currently modifiability, availability, security, and performance) and is aimed at locating and analyzing tradeoffs in a software architecture, for these are the areas of highest risk in an architecture.

We have developed a *method* so that the analysis is repeatable and transitionable. Having a structured method helps ensure that the right questions regarding an architecture will be asked *early*, during the requirements and design stages when discovered problems can be solved cheaply. It guides users of the method—the stakeholders—to look for conflicts and for resolutions to these conflicts in the software architecture.

In the past much of software engineering practice has paid lip-service to quality attributes in designing software architectures, but has done little to ensure that these quality attributes are satisfied by the design. Recent efforts on cataloguing the implications of using design patterns and architectural styles contribute, frequently in an informal way, to ensuring the quality of a design [4]. More formal efforts also exist to ensure that quality attributes are addressed. These consist of analyses in areas such as performance evaluation [8], Markov modeling for availability [5], and inspection and review methods for modifiability [6].

But these techniques, if they are applied at all, are typically applied in isolation and their implications are considered in isolation. This is dangerous. It is dangerous because *all* design involves tradeoffs and if we simply optimize for a single quality attribute, we stand the chance of ignoring other attributes of importance. Even more significantly, if we do not analyze for multiple attributes, we have no way of understanding the tradeoffs made in the architecture—places where improving one attribute causes another one to be compromised.

It is important to clearly state what the ATAM is and is not. The ATAM is meant to be a risk mitigation method; a means of detecting areas of potential risk within the architecture of a complex software intensive system. This has several implications: 1) the ATAM can be done early in the software development life cycle; 2) it can be done inexpensively and quickly (because it is assessing architectural design artifacts); 3) it does not need to produce detailed analyses of any measurable quality attribute of a system (such as latency or mean time to failure) to be successful but instead identifies *trends* where some architectural parameter is correlated with a measurable quality attribute of interest. This final point is crucial in understanding the goals of the ATAM; we are not interested in precisely characterizing any quality attribute. That would be pointless at an early stage of design. 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 devoted more of our design, analysis, and prototyping energies on such decisions.

This paper will describe the method (more details can be found in [1] and [7]), will present an example analysis, and

will discuss the implications of the ATAM.

2 ATAM STEPS

The steps of the method are as follows:

- Step 0 - *Planning/Information exchange*: in this step we hold a meeting where we describe the method to the stakeholders, set expectations, learn about the stakeholders' main quality goals for the system, and see the architect's initial presentation of the architecture and initial set of scenarios.
- Step 1 - *Scenario brainstorming*: This step begins the ATAM proper. We gather the important system stakeholders and facilitate the brainstorming of scenarios of uses of the system, faults of the system, and anticipated changes to the system. During this phase the analysts add or augment the scenarios based upon the quality attributes under review, their experience, and their need for additional insight into the architecture.
- Step 2 - *Architecture presentation*: the architecture is presented in detail and the most important and illustrative normal usage scenarios are mapped onto the architecture, to aid in understanding the system and, in particular, how data and control flow through it. The analysts attempt to identify and probe the ramifications of architectural styles here.
- Step 3 - *Scenario coverage checking*: we use a set of standard quality attribute-specific questions to ensure proper coverage of an attribute by the scenarios. In particular, we look to see if boundary conditions have been covered.
- Step 4 - *Scenario grouping and prioritization*: the stakeholders vote on the scenarios that are of highest concern for them. During this phase they can suggest grouping scenarios. After the voting is complete, we determine a cutoff point at 10-15 scenarios.
- Step 5 - *Map high priority scenarios onto architecture*: in this step the architect walks through each high-priority attribute specific scenario, showing how it affects the architecture (e.g. for modifiability) and how the architecture responds to it (e.g. for quality attributes such as performance, security and availability).
- Step 6 - *Perform quality attribute-specific analysis*: the architect guides the analysis showing why the architecture meets the attribute-specific requirements, as illuminated by the scenarios of interest. The analysts build models of each quality attribute based upon the architect's information. By systematically manipulating the input parameters to the model, the analysts determine sensitivity points—parameters in the architecture to which some measurable quality attribute is highly correlated—are determined. In this way, the architectural parameters or elements to which this scenario is sensitive are identified. For example, if end-to-end latency is highly sensitive to the size of a queue, the queue's size is noted as an architectural sensitivity point.

  During this step, the analysts may discover that the existing architecture is inadequate. As a consequence, architectural alternatives may be suggested, and these will feed into the action plan developed in step 8.
- Step 7 - *Identify trade-off points*: to find tradeoffs we locate all important architectural elements in which multiple sensitivities exist. For example, the number of copies of a database might be a sensitivity point for both availability and performance.
- Step 8 - *Consolidate findings and develop action plan*: this plan is a set of recommendations for improving the architecture in the light of the analysis findings. Additionally, we might ask for more supporting documentation such as: more architectural information, scenarios, environmental information, platform information, details about constraints, or justification for requirements.

After step 8 we might decide to modify the architecture, which necessitates a return to step 1, wherein we analyze the consequences of our decisions. Typically step 0 takes place on a single day, steps 1 through 4 take place on a second day, and steps 5 through 8 take place on a third day. The impact on the project team being analyzed is small—typically from 10 to 40 person days of their time in projects that measure from 10 to 200 person years.

In addition to these steps, we are developing a handbook at the SEI:[1] a set of materials that accompanies the evaluation that describe many of the evaluation artifacts. For example, we have a handbook section that describes attribute based architectural styles (ABASs)—styles that are focussed on addressing performance, modifiability, security, and availability along with accompanying analytical frameworks. We also have a handbook section, that is used heavily in steps 3 and 6, with a set of quality attribute-specific questions that aid us in probing the architecture. For example, when building a performance model of some portion of the system, we ask questions such as the following to elicit more information about a scenario:

- What event starts the scenario. For example, is it a message arrival, keystroke, mouse click, state change, the passage of time,...?
- How often does this initiating event occur (e.g. periodically with a fixed rate, stochastically with a known mean)?
- What is the performance requirement (e.g. hard deadline, soft deadline, throughput, average-case response time)? Quantify if possible (e.g. hard deadline of 100 ms).
- What is the consequence of not meeting the performance requirement (e.g. catastrophe, inconvenience, annoyance, system failure and reboot).

By following a standard set of quality-specific questions, we elicit the information needed to analyze that quality in a predictable, repeatable fashion [10]. In addition to attribute-specific elicitation questions we have a set of questions that aid us in gathering the information needed to build an

1. The ATAM web site—http://www.sei.cmu.edu/ata/ata_init.html—contains additional background materials on performing these evaluations.

analytic model of the quality. For example, in gathering performance information we elicit information about resource usage and contention: the raw data needed to build an analytic performance model [8].

3 THE RATIONALE FOR ATAM

The ATAM is a spiral model [3] of *design* and *analysis*; in our view one cannot be done without the other. A design is what you analyze and an analysis tells you how to go about refining a design. Thus, this process is not simply a front-end gate that a system passes through on its journey from requirements to fielded system. Analyses must live with a design throughout a system's lifetime, so that the system is built appropriately and maintained correctly. In addition, the ATAM helps drive design: it identifies areas of risk and helps to plan for risk mitigation. It also drives the documentation of the architecture, as we will show in our example.

The ATAM draws its inspiration and techniques from three areas: the Software Architecture Analysis Method (SAAM) [6], quality attribute communities, and the notion of architectural styles [11]. The ATAM is intended to analyze an architecture with respect to its quality attributes, not its functional correctness. Although this is the ATAM's focus, there is a problem in operationalizing this focus: we (and the software engineering community in general) do not understand quality attributes well: what it means to be "open" or "interoperable" or "secure" or "high performance" changes from system to system and from community to community. So, we turn to scenarios as a means of operationalizing the analysis of quality attributes. The focus of the SAAM is the use of scenarios for architectural modifiability evaluation. Scenarios provided a vehicle for concretizing modifiability; they represent specific examples of current and future uses of a system. The future uses typically imply modifications against which the architecture can be assessed, thereby transforming the abstract notion of modifiability into concrete modification scenarios.

Performing the ATAM has taught us that scenarios are also the driving force in understanding run-time qualities (such as performance or availability). This is because scenarios specify the kinds of operations over which performance needs to be measured, or the kinds of failures the system will have to withstand.

The ATAM also builds on the knowledge bases associated with quality attributes. We organize this knowledge into what we call *attribute models*. We augment scenarios with attribute-centric questions such as those shown in the previous section, based upon an analytic model of each quality attribute. Building and analyzing an attribute model helps to answer the following three questions: 1) What are the measurable or observable manifestations of the attribute? 2) What are the attribute-relevant stimuli or events to which the architecture must respond? 3) What are the characteristics of the architecture that affect the observable manifestation?

Finally, the ATAM builds on the concept of architectural styles [11]. "An architectural style is a description of component types and a pattern of their run-time control and/ or data transfer. A style can be thought of as a set of constraints on an architecture—constraints on component types and their interactions—and these constraints define a set or family or architectures that satisfy them" [2]. The ATAM uses a particular specialization of architectural styles, ABASs.

An ABAS is an architecture style in which the constraints focus on component types and patterns of interaction that are particularly relevant to quality attributes such as performance, modifiability, security or availability. ABASs aid architecture evaluation by focusing the stakeholders' attention on the patterns that dominate the architecture. This focus is accomplished by highlighting the attribute-specific questions associated with the pattern, and by placing the answers to these questions into an *analytic framework*. For example, if an architecture used a collection of interacting processes, this could be recognized as a performance ABAS. The questions associated with this performance ABAS would probe important architectural parameters such as the priority of the processes, estimates of their execution time, places where they synchronize, queuing disciplines, etc.; information that relevant to understanding the performance of this style. The answers to these questions then feed into an explicit analytic framework such as rate monotonic analysis for performance [8].

4 AN EXAMPLE EVALUATION: THE BCS

We have now performed 5 ATAM-based evaluations (2 internal and 3 external). The example that we are presenting has had some of the details changed to protect the identity and intellectual property of the customer and contractor, but the architectural issues that we uncovered are not materially affected by these changes.

We shall call the system BCS (Battlefield Control System). This system is to be used by Army battalions to control the movement, strategy, and operations of troops in real time in the battlefield. This system is currently being built by a contractor, based upon government furnished requirements. These state that there is a Commander who commands a set of Soldiers and equipment, including many different kinds of weapons and sensors. The system needs to interface with numerous other systems that feed it commands and intelligence, and collect its status with respect to its missions.

Step 0 - Planning/Information Exchange

During the initial "pre-meeting" we presented the method, the contractor presented the architecture, and the contractor and customer described their initial set of scenarios. As a result of this meeting additional architectural documentation was requested. As is often the case in evaluating architectures, the initial documentation that was produced was far too vague to support any analysis, consisting of high level data flows and divisions of functionality that had no clear realization in software. Thus, additional information was requested, in the form of questions, to address the gaps in the original documentation produced by the contractor:

- what is the structure of the message handling software (i.e. how the functionality is broken down in terms of

modules, functions, APIs, layers, etc.)?

- what facilities exist in the software architecture (if any) for self-testing and monitoring of software components?
- what facilities exist in the software architecture (if any) for redundancy, liveness monitoring, failover, and how data consistency is maintained (so that one component can take over from another and be sure that it is in a consistent state with the failed component)?
- what is the process and/or task view of the system, including mapping of these processes/tasks to hardware and the communication mechanisms between them?
- what functional dependencies exist among the software components (often called a "uses" view)?
- what data is kept in the database (which was mentioned by one of your stakeholders), how big it is, how much it changes, and who reads/writes it?
- what is the anticipated frequency and volume of data being transmitted among the system components?

Between the first and second days of the evaluation the contractor answered many of these questions and produced substantially more complete, more usable architectural documentation. This formed the basis for scenario mapping and evaluation in the next steps of the ATAM.

Step 1 - Scenario Brainstorming

A scenario represents a use of—a stimulus to—the BCS architecture, applied not only to determine if the architecture meets a functional requirement, but also (and more significantly) for prediction of system qualities such as performance, availability, modifiability, and so forth. Direct scenarios are those that are satisfied by the architecture through the execution of the system. Direct scenarios that correspond to requirements previously addressed in the design process will not be surprising to the stakeholders, but will increase their understanding of the architecture and allow systematic investigation of architectural qualities such as performance and availability. An indirect scenario is one that requires a modification to the architecture to satisfy it; indirect scenarios are central to the measurement of the degree to which an architecture can accommodate evolutionary changes that are important to the stakeholders. The cumulative impact of indirect scenarios on an architecture measure its suitability for ongoing use throughout the lifetime of a family of related systems.

The scenario elicitation process allows stakeholders to *contribute* scenarios that reflect their concerns and understanding of how the architecture will accommodate their needs. A particular scenario may, in fact, have implications for many stakeholders: for a modification, one stakeholder may be concerned with the difficulty of a change and its performance impact, while another may be interested in how the change will affect integrability of the architecture.

Scenarios were collected by a round-robin brainstorming activity in which no criticism and little or no clarification was provided. Table 1 shows a few of the 40 scenarios that were elicited in the first day of the ATAM evaluation[2].

Table 1: Sample Scenarios for the BCS Evaluation

Scenario	Scenario Description
1	Same information presented to user, but different presentation (location, fonts, sizes, colors, etc.).
2	Additional data requested to be presented to user.
3	User requests a change of dialog.
4	An new device is added to the network, e.g. a location device that returns accurate GPS data.
5	An existing device adds additional fields that are not currently handled to existing messages.
6	Map data format changes.
7	The time budget for initialization is reduced from 5 minutes to 90 seconds.
8	Modem baud rate is increased by a factor of 4.
9	Operating system changes to Solaris.
10	Operating schedule is unpredictable.
11	Can a new schedule be accommodated by the OS?
12	Change the number of Soldier nodes from 25 to 35.
13	Change the number of simultaneous missions from 3 to 6.
14	A node converts from being a Soldier/client to become a Commander/server.
15	Incoming message format changes.

Step 2 - Architecture Presentation

As mentioned above, the architectural documentation covered several different views of the system: a dynamic view, showing how subsystems communicated; a set of message sequence charts, showing run-time interactions; a system view, showing how software was allocated to hardware; and a source view, showing how components and subsystems were composed of objects. For the purpose of this presentation, we will just show the highest level system

2. These scenarios have been cleansed of proprietary specifics, but their spirit is true to the original.

view of the architecture, using the notation from [2].[3]

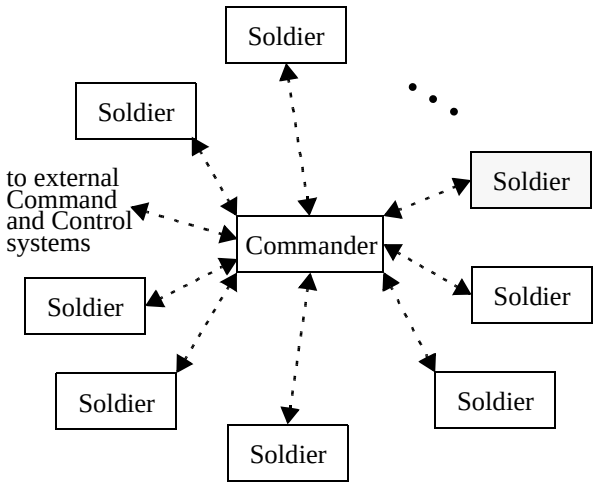

Figure 1: System Architecture of the BCS

This system architecture, shown in Figure 1, illustrates that the Commander is central to the system; it acts as a server and the Soldier nodes are its clients, making requests of it and updating the server's database with their status. Interaction between the client and server is only through encrypted messages sent via a radio modem; neither subsystem controls the other. Note also that the radio modem is a shared communication channel: only one node can be broadcasting at any moment.

Step 3 - Scenario Coverage Checking
At this stage in the analysis we had a set of architectural documentation but little insight into the way that the architecture worked to accomplish the BCS's mission. So we used a set of quality-attribute specific questions from our handbook to flesh out our understanding of the architecture. These questions probed both the normal operation of the system and its boundary conditions. For example:

- For what functions of the system is performance **not** important?
- For those functions for which performance is **not** important what is the consequence of extremely long response times or extremely low throughput?
- How is performance impacted by scaling up the workload?

By itself, Figure 1 tells us little about the system. However, when illuminated by a small number of scenarios and attribute-specific questions, this view became the focus for availability (or, in the customer's terms survivability) and performance analyses.

The next step in the ATAM was to select scenarios for more detailed consideration.

3. In this notation, rectangles represent processors and dashed lines represent data flow.

Step 4 - Scenario Grouping and Prioritization
Some of the scenarios presented in Table 1 have obvious commonalities. The goal of the next step—scenario grouping—is to identify related scenarios, where "related" means that the scenarios are expected to have similar effects on the architecture.

Scenario groups were proposed and defended by the BCS stakeholders. Prioritization of the grouped scenarios allows the most important scenarios to be addressed within the limited amount of time (and energy) available for the evaluation. Here, "important" is defined entirely by the stakeholders. The prioritization is accomplished by giving each stakeholder a fixed number of votes; 30% of the total number of scenarios has been determined to be a useful heuristic. Thus, for the BCS, each stakeholder was given 12 votes that they use to vote for scenarios in which they were most interested. A stakeholder may allocate as many or as few votes as they wish to each scenario. Typically, the resulting totals will provide an obvious cutoff point; 10-15 scenarios are the most that can be considered in a normal one-day session; for the BCS a natural cutoff occurred at 12 scenario groups. Some negotiation is appropriate in choosing which scenarios to consider; a stakeholder with a strong interest in a particular scenario can argue for its inclusion, even if it did not receive a large number of votes in the initial prioritization.

Step 5 - Map High Priority Scenarios Onto Architecture
In the ATAM process, once a set of scenarios has been chosen for consideration, these scenarios are "mapped" onto the architecture. In the case of a scenario that implies a change to the architecture, the architect demonstrates how the scenario would affect the architecture in terms of the changed, added, or deleted components, connectors, and interfaces. For the case in which the architecture, as designed, is able to "execute" the scenario, the architect traces this execution path through the relevant components and connectors.

Stakeholder discussion is important here to elaborate the intended meaning of a scenario description and to discuss how the mapping is or is not suitable from their perspective. The mapping process also illustrates weaknesses in the architecture and its documentation.

For the BCS, each of the high priority scenarios was mapped onto the appropriate architectural view. For example, when a scenario implied a modification to the architecture, the ramifications of the change were mapped onto the source view, and scenario interactions were identified as sensitivity points. For availability and performance, failure and usage scenarios were mapped onto run-time and system views of the architecture, and models were built of latency and availability based upon the information elicited by these mappings.

Step 6 - Perform Quality Attribute Specific Analysis
A sensitivity point for an attribute is defined as a parameter in the architecture to which some measurable attribute is highly correlated; small changes in such parameters are likely to have significant effects on the measurable behavior

of the system. For this reason we must focus our attention on these points as they pose the highest risks to the system's success, particularly as the system evolves.

We find sensitivity points by building models of a quality attribute. We build a collection of formal analytic models such as RMA models for performance, and Markov models for availability, and informal models such as that used in SAAM for modifiability. These models are frequently quite simple initially. Once we build these models we experiment with their parameters until we determine which ones have substantial effects on a measurable attribute such as latency or throughput or mean time to failure. The point of the model building is twofold: to find sensitivity points (not to precisely characterize a measurable attribute, for it is typically too early in the development process to do this with any rigor); and to gain insight into the architecture, via the elicitation process that model building requires.

For the BCS system we realized through the ATAM process that three quality attributes were the major architectural drivers for overall system quality: availability, modifiability, and performance. Hence we can say that system quality (Q_S), is a function f of the quality of the modifiability (Q_M), the availability (Q_A), and the performance (Q_P):

$$Q_S = f(Q_M, Q_A, Q_P)$$

The next sections will describe the analyses of each of these qualities.

Availability
A key quality attribute for the BCS was determined to be its steady-state availability, i.e.

$$Q_A = g(\text{the fraction of time that the system is working})$$

The system is considered to be working if there is a working Commander and any number of Soldier nodes. When the Commander fails the system has failed. Provisions have been made in the BCS architecture, however, to turn a Soldier into a Commander, i.e. converting a client into a server. The repair time for the system is the time to turn a Soldier node into the Commander and thus restore the system to operation. Failure of the Commander is detected via human-to-human communication.

As identified by our scenarios, the key stimulus to model for the system is the failure of a node (Commander or Soldier node) in the system due to an attack, hardware failure, or software failure. The architecture for BCS currently statically specifies an Commander and a single backup selected from among the Soldier nodes, as indicated by the shaded Soldier node in Figure 1. In the existing design *acknowledged* communication takes place between the Commander and the backup Soldier node to allow the backup to maintain a state of readiness in case of failure of the Commander. Upon failure of the Commander, the backup takes over as Commander, converts from being a client to a server (as indicated by Scenario 14). In particular, there is no provision to have one of the surviving Soldier nodes promoted to become the current backup. So, we can refine our characterization of the system's availability as follows:

$$Q_A = g(\lambda_C, \mu_C, \mu_B)$$

That is, system availability is primarily affected by the failure rate of the Commander (λ_C), the repair rate of the Commander (μ_C, the time required to turn the backup into the Commander). The availability might also be affected by the repair rate of the backup, but in the BCS as it is currently designed, the repair rate of the backup (μ_B) is 0 since there is no provision for creating additional backups.

However, by building a simple model of the system's availability, we can determine the effects on the system of changing the repair rate of the backup. Specifically, we can determine the amount of time required for a new backup to enter a readiness state (i.e. where it could quickly become the Commander). This would require a change to the architecture.

An alternative architecture could allow multiple (perhaps all) Soldier nodes to monitor the Commander-to-backup communication and thus maintain a higher level of readiness. And these additional backups could either acknowledge communication with the Commander (requesting resends of missed packets) or could be silent receivers of packets, or some mixture of these schemes (i.e. the top *n* backups acknowledge receipt of packets, and the remaining *m* backups are passive). In the case where packets are not acknowledged, the state of the backups database would increasingly drift from that of the Commander and if one of these backups is called upon to become the Commander, it would need to engage in some negotiation (with the external systems and/or the other Soldier nodes) to complete its database.

Thus, there are three considerations for changing the BCS architecture to improve the data distribution to the backups:

- a backup could be an "acknowledging backup", which is kept completely synchronized with the Commander
- a backup might be only a "passive" backup and not ask for re-sends when it misses a message; this implies that it has the means for determining that it has missed a message (such as a message numbering scheme)
- a backup, when it becomes the new Commander, or when it becomes an "acknowledging backup", could request any missed information from the upper level Command and Control systems and/or the other Soldier nodes.

The system does not need to choose a single option for its backups. It might have *n* acknowledging backups and *m* passive backups. Assuming that we have no control over the failure rate of the Commander, then the true sensitivities in this system with respect to availability are functions of the repair rates of the Commander and backups, which are themselves functions of the numbers of acknowledging and passive backups. Now we have a usable description of the architectural sensitivities—a correlation between some architectural parameter and a measurable attribute:

$$Q_A = g(n, m)$$

What are the issues in the choice of the number of backups to keep and whether they acknowledge communications or not?

We consider this issue in terms of the failure and recovery rates of the system under the various options. Clearly, the availability of the system increases as the number of backups is increased, because the system can survive multiple failures of individual nodes without failing its mission. The availability of the system is also increased by increasing the number of acknowledging backups, for two reasons: 1) acknowledging backups can be ready to assume the responsibilities of an Commander much more quickly, because they do not need to negotiate with other nodes for missed information, and; 2) having more acknowledging backups means that there will not be an identifiable communication pattern between the Commander and the single backup, as there is currently, which means that the probability of two accurate incoming mortars disabling the system is reduced.

However, as the number of acknowledging backups is increased, the performance of the system is impacted, as each of these acknowledgments incurs a small communication overhead. Collectively, this overhead is significant, because as we will discuss next, communication latency is the major contributor to overall system latency for BCS.

Performance

By building a simple performance model of the system and varying the input parameters to the model (the various processing times and communication latencies), it became clear that the slow speed of radio modem communication between the Commander and the Soldiers (9600 baud) was the single important performance driver for the BCS. The performance measure of interest—average latency of client-server communications—was found to be insensitive to all other architectural performance parameters (e.g. the time for the system to update its database, or to send a message internally to a process, or to do targeting calculations). But the choice of modem speed was given as a constraint, and so our performance model was focused on capturing those architectural parameters that affected message sizes and distributions.

We begin by identifying the scenarios, from among all those considered, that have performance implications and the communication requirements implied in each scenario. For example, we considered three scenario groups (not all of which appear in Table 1) when building our performance model:

A) Scenarios 14, 23, 25, 29 (turning a Soldier node into a backup): a switchover requires that the backup acquires information about all missions, updates to the environmental database, issued orders, current Soldier locations and status, and detailed inventories from the Soldiers.

B) Scenarios 26, 27 (regular, periodic data updates to the Commander): various message sizes and frequencies.

C) Scenarios 13, 20: Increasing number of weapons from 30 to 60 or missions from 3 to 6.

We created performance models of each of these scenario groups. For the purposes of illustration in this paper, we will only present the performance calculations for scenario group A), the conversion from Soldier backup to Commander.

After determining that a switchover is to take place the Soldier backup will need to download the current mission plans and environmental database from the external command and control systems. In addition, the backup needs the current locations and status of all of the remaining Soldiers, inventory status from the Soldiers, and the complete set of issued orders.

A typical calculation of the performance implications of this scenario group will take into account the various message sizes needed to realize the scenario, the 9600 baud modem rate (which we equate to 9600 bits/second), and the fact that there are a maximum of 25 Soldiers per Commander (but since one is now being used as a Commander, the number of Soldier nodes in these calculations is 24):

Downloading mission plans:
280 Kbits / 9.6 Kbits/second ≅ 29.17 seconds.

Updates to environmental database:
66 Kbits / 9.6 Kbits/second ≅ 6.88 seconds.

Acquiring issued orders:
24 Soldiers * (18 Kbits/9.6 Kbits/second) = 45.0 seconds.

Acquiring Soldier locations and status:
24 Soldiers * (12 Kbits/9.6 Kbits/second) = 30.0 seconds.

Acquiring inventories:
24 Soldiers * (42 Kbits/9.6 Kbits/second) = 105.0 seconds.

Total ≅ 216.05 seconds for Soldier to become backup

Note that, since the radio modem is a shared communication channel, no other communication can take place while a Soldier/backup is being converted to a Commander.

There was no explicit requirement placed on the time to switch from a Commander to a backup. However, there was an initialization requirement of 300 seconds which we will use in lieu of an explicit switchover time budget. If we assume that the 280K bits in the mission plan file contains the 3 missions in the current configuration, then doubling the number of missions (scenario 13) would imply doubling the mission message from 280K bits to 560K bits and the transmission time would increase by almost 30 seconds, still meeting the time budget. If, on the other hand, the number of Soldiers increases to 35 (scenario 12), the total time will increase by about 90 seconds, which would not meet the time budget.

Keeping each backup in a state of high readiness requires that they become acknowledging backups, or for a lower state of readiness they can be kept as passive backups. Both classes of backups require periodic updates from the Commander. From an analysis of scenario group B), we have calculated that these messages average 59,800 Kbits every 10 minutes. Thus, to keep each backup apprised of the state of the Commander requires 99.67 bits/second, or approximately 1% of the system's overall communication

bandwidth. Acknowledgments and resends for lost packets would add to this overhead. Given this insight, we can characterize the system's performance sensitivities as follows:

$$Q_P = h(n, m, CO)$$

That is, the system is sensitive to the number of acknowledging backups (*n*), passive backups (*m*), and other communication overhead (*CO*). The main point of this simple analysis is to realize that the size and number of messages to be transmitted over the 9600 baud radio modem is important with respect to system performance and hence availability. Small changes in message sizes, or frequencies can cause significant changes to the overall throughput of the system. These changes in message sizes may come from changes imposed upon the system

Modifiability
Scenarios 1, 2, 3, 5, 6, and 31 were mapped onto a source view of the architecture to understand their implications. One component of the architecture appeared—Message Manager—appeared to have a high level of scenario interaction. What this means is that the satisfaction of many different indirect scenarios (in this case scenarios 2, 3, 5, 6, and 31, which together represent almost half of the indirect scenarios that we considered) required a modification to the Message Manager. While this does not, in itself, prove that there is a problem with Message Manager, it does indicate that this component will potentially be a sensitivity point in the architecture: a location of many changes and hence of high potential complexity.

Step 7 - Identify Tradeoff Points
We have identified three sensitivities in the BCS system, and two of these are affected by the same architectural parameter: the amount of message traffic that passes over the shared communication channel employed by the radio modems, as described by some functions of *n* and *m*, the numbers of acknowledging and passive backups. Recall that availability and performance were characterized as:

$$Q_A = g(n, m)$$

and

$$Q_P = h(n, m, CO)$$

These two parameters control the tradeoff point between the overall performance of the system, in terms of the latency over its critical communication resource, and between the availability of the system in terms of the number of backups to the Commander, the way that the state of those backups is maintained, and the negotiation that a backup needs to do to convert to a Commander. To determine the criticality of the tradeoff more precisely, we can prototype or estimate the currently anticipated message traffic and the anticipated increase in message traffic due to acknowledgments of communications to the backups. In addition, we would need to estimate the lag for the switchover from Soldier to Commander introduced by not having acknowledged communication to the Solder backup nodes. Finally, all of this increased communication needs to be considered in light of the performance scalability of the system (since communication bandwidth is the limiting factor here).

One way to mitigate against the communication bandwidth limitation is to plan for new modem hardware with increased communication speeds. Presumably this means introducing some form of indirection into the modem communications software—such as an abstraction layer for the communications—if this does not already exist. This possibility was not probed during the evaluation.

While this tradeoff might seem obvious, given the presentation here, it was not so. The contractor was not aware of the performance and availability implications of the architectural decisions that had been made. In fact, in our initial pre-meeting, not a single performance or availability scenario was generated by the contractor; these simply were not among their concerns. The contractor was worried about the modifiability of the system, in terms of the many changes in message formats that they expected to withstand over the BCS's lifetime. However, the identified tradeoff affected the very viability of the system. If this tradeoff was not carefully reasoned about, it would affect the system's ability to meet its most fundamental requirements.

5 RESULTS OF THE BCS ATAM

The ATAM that we performed on the architecture for the BCS revealed some potentially serious problems in the documentation of the architecture, the clarity of its requirements, its performance, its availability, and a potential architectural tradeoff. We will briefly summarize each of these problems in turn.

Documentation
The documentation provided at the inception of this project was minimal: two pages of diagrams that did not correspond to software artifacts in any rigorous way. This is, in our experience, typical, and is the single greatest impediment to having a productive architectural evaluation. Having a pre-meeting and having the opportunity to request additional documentation from the contractor made the evaluation successful.

As a result of our interaction with the BCS contractor team, substantially greater and higher quality architectural documentation was produced. This improved documentation became the basis for the evaluation. And the improvement in the documentation was identified by management as a major success of the ATAM process, even before we presented any findings.

Requirements
One benefit of doing any architectural evaluation is increased stakeholder communication, resulting in better understanding of requirements. Frequently, new requirements surface as a result of the evaluation. The BCS experience was typical, even though the requirements for this system were "frozen" and had been made public for over 2 years.

For example, in the BCS the only performance timing requirements were that the system be ready to operate in 5 minutes from power-on. In particular, there were no timing requirements for other specific operations of the system,

such as responding to a particular order, or updating the environment database. These were identified as lacking by the questions we asked in building the performance model.

Furthermore, there was no explicit switchover requirement, i.e. the time that it takes for a Soldier to turn itself into a Commander is not identified as a requirement. This requirement surfaced as a result of building the availability model.

In addition, there was no stated availability requirement. Two well aimed hits, or two specific hardware failures and the system, as it is currently designed, is out of commission. This was seen, by the stakeholders, as a major oversight in the system's design.

Sensitivities and Tradeoffs

As identified above, the Message Manager component presents a possible scenario interaction problem with respect to modifiability. We identified the need to explore the substructure of the Message manager and additional indirect scenarios need to be generated and mapped onto the architecture to better understand the implications of anticipated changes on the complexity of the software architecture.

The most important tradeoff identified for the BCS was the communications load on the system, as it was affected by various information exchange requirements and availability schemes. The overall performance and availability of the system is highly sensitive to the latency of the (limited and shared) communications channel. Not only should the current performance characteristics be modeled, but also the anticipated performance changes in the future as the system scales in its size and scope.

Architectural Problems

In addition to the sensitivities and tradeoffs, in building the models of the BCS's availability and performance, we discovered a serious architectural weakness that had not been previously identified: there exists the possibility of an opposing force identifying the distinctive communication pattern between the Commander and the backup and thus targeting those nodes specifically. The Commander and backup exchange far more data than any other two nodes in the system. This identification can be easily done by an attacker who could discern the distinctive pattern of communication between the Commander and (single) backup, even without being able to decrypt the actual contents of the messages. Thus, it must be assumed that the probability of failure for the Commander and its backup increases over the duration of a mission under the existing BCS architecture. This was a major architectural flaw that was only revealed *because* we were examining the architecture from the perspective of multiple quality attributes simultaneously. This flaw is, however, easily mitigated by assigning multiple backups, which would eliminate the distinctive communication pattern.

Step 8 - Consolidate Findings and Develop Action Plan

As described above, after the evaluation the contractors were sent a detailed report of the ATAM. In particular, they were alerted to the potential modifiability, performance, and availability problems in the BCS. As an action plan, the contractor was furnished with three architectural possibilities for adding multiple Soldier backups and keeping them more or less synchronized with the Commander. Each of these architectural deltas had their own ramifications that needed to be modeled and assessed. And the alternatives will be chosen different depending on other architectural choices that affect the performance model, such as the radio modem speed.

The point of this paper is not to show which alternative the contractor chose, for that is relatively unimportant and relied on their organizational and mission-specific constraints. The point here is to show that the process of performing an ATAM on the BCS raised the stakeholders' awareness of critical issues, focused design activity in the areas of highest risk, and caused a major iteration within the spiral process of design and analysis.

6 LESSONS LEARNED WITH ATAM

Although our ATAM experience base is still small, it builds upon a much larger experience base garnered in doing SAAM evaluations. From this experience, we have observed a number of issues worth noting.

First, since this is a scenario-based method and scenarios come from the system's stakeholders, dealing with stakeholders successfully is crucial. However, it is not always easy. The goals of the architectural evaluation must be made clear, because we, as external evaluators, are always regarded with some suspicion and we are taking time out of the stakeholder's schedules. Getting the stakeholders to "buy-in" to the process is essential and this means making them understand the steps of the ATAM, why *these* steps are important, and why they need to occur in *this* order. For example, the stakeholders need to be focused on the scenarios that represent *critical* uses of and anticipated changes to the system.

Another issue that we have noted from performing ATAM (and SAAM) evaluations is that getting consensus on the right set of scenarios works differently in different organizations. So, we must be sensitive to the different styles of organizations. Some organizations are democratic, others are strictly top-down hierarchies, some have centralized decision making while others are distributed, some organizations have openly antagonistic sub-groups, others are relatively harmonious. These different styles are important because the success of the ATAM rests squarely on eliciting the right scenarios and prioritizing them correctly.

Having good architectural documentation is crucial to the success of any architectural evaluation method; the ATAM is a garbage-in-garbage-out process. If the documentation is inadequate, we end up spending most of the time redocumenting the architecture, rather than analyzing it. "Good" architectural documentation means having multiple views of the architecture, as Kruchten pointed out [9]. We like to have: source, system, and dynamic views, annotated with scenarios and message sequence charts. And we need to understand the mapping between these views, so that we can

determine how a change in one view will affect the representations and analytic models in another view.

The ATAM is not meant to provide definitive answers about the performance or security or modifiability or availability of the system being built. It is meant to focus design activity, analysis activity, and stakeholder attention. All of this focus is toward the goal of identifying sensitivity points in the architecture, which are the sites of potentially high risk with respect to the system's ability to meet its current and future requirements. This knowledge is not intuitive; it requires a method that forces the stakeholders to state what is important to them, both at the moment and in the future. It forces the architects to state unambiguously how the system will satisfy the stakeholders' scenarios. And this allows analysts to build models of the system's critical properties.

Building models is its own benefit: we often learn more by eliciting the information needed by the models and then building the models than analyzing the models. And we don't expect precise answers from these models: the time in which an ATAM is performed is typically too early in the development process for such analyses to be trusted, or for the effort in creating detailed analyses to be justified. We build the models to elicit information about the system and to find trends in the system that will affect its evolution.

Our final observation culled from our experience with the ATAM is that we have noticed that the iteration of design and analysis really works. Each time that we go through this process—collect scenarios, ask questions, ask to have the architecture presented to us, build models, and critique the architecture—the architecture and the way that it is documented changes (and, in our opinion and the opinions of the project teams, changes for the better). The ATAM is truly working as a spiral model of analysis and design.

7 REFERENCES

1. Barbacci, M., Carrière, J., Kazman, R., Klein, M., Lipson, H., Longstaff, T., Weinstock, C., "Steps in an Architecture Tradeoff Analysis Method: Quality Attribute Models and Analysis", CMU/SEI-97-TR-29, Software Engineering Institute, Carnegie Mellon University, 1997.
2. Bass, L., Clements, P., Kazman, R., *Software Architecture in Practice*, Addison Wesley, 1998.
3. Boehm, B., "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, 21(5), May 1988, pp. 61-72.
4. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., *Pattern-Oriented Software Architecture*, Wiley, 1996.
5. Iannino, A., "Software Reliability Theory", *Encyclopedia of Software Engineering*, Wiley, 1237-1253.
6. Kazman, R., Abowd, G., Bass, L., Clements, P., "Scenario-Based Analysis of Software Architecture", *IEEE Software*, Nov. 1996, 47-55.
7. Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carriere, J., "The Architecture Tradeoff Analysis Method", *Proceedings of ICECCS '98*, (Monterey, CA), August 1998, 68-78.
8. Klein, M., Ralya, T., Pollak, B., Obenza, R., Gonzales Harbour, M., *A Practitioner's Handbook for Real-Time Analysis*, Kluwer Academic, 1993.
9. Kruchten, P., "The 4+1 View Model of Architecture", *IEEE Software*, 12(6), Nov. 1995.
10. Maranzano, J., *Best Current Practices: Software Architecture Validation*, AT&T Technical Report, 1993.
11. Shaw, M., Garlan, D., *Software Architecture: Perspectives n an Emerging Discipline*, Prentice Hall, 1996.