Technical Report
CMU/SEI-96-TR-025
ESC-TR-96-025

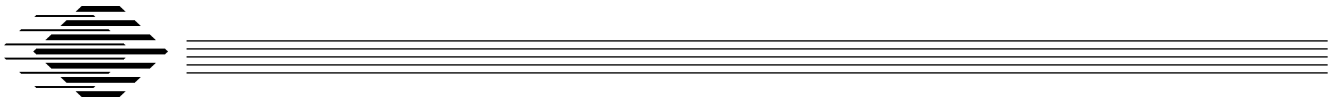
Recommended Best Industrial Practice for Software Architecture Evaluation

Gregory Abowd, Georgia Institute of Technology
Len Bass, SEI
Paul Clements, SEI
Rick Kazman, SEI
Linda Northrop, SEI
Amy Zaremski, SEI

January 13, 1997

Technical Report
CMU/SEI-96-TR-025
ESC-TR-96-025
January 1997

Recommended Best Industrial Practice for Software Architecture Evaluation



Gregory Abowd, Georgia Institute of Technology
Len Bass, SEI
Paul Clements, SEI
Rick Kazman, SEI
Linda Northrop, SEI
Amy Zaremski, SEI

Product Line Systems Program

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Pittsburgh, PA 15212.
Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: (703) 767-8222 *or toll-free in the U.S. — 1-800 225-3842).

Table of Contents

List of Tables

Acknowledgments

The authors of this report thank the workshop participants listed in the appendix and their sponsoring organizations. We also thank Jennifer Bitters and Barbara Tomchik, assistants at the SEI, who helped host the workshops.

Recommended Best Industrial Practice for Software Architecture Evaluation

Abstract: Architectural decisions have a great impact on the consequent quality of software systems. As a result, it is important to evaluate how a software architecture meets its quality demands. Though much focus has been placed on modeling and describing the software architecture as a design artifact, we found that relatively little is known about the current experience with software architecture evaluation.

This report details the results of two workshops on software architecture evaluation, held at the Software Engineering Institute (SEI) on November 9-10, 1995 and May 9-10, 1996. The purpose of the workshops was to determine the state of industrial practice in the evaluation of software architectures with respect to a set of desired quality attributes, and to uncover recommendations for best practices. In this report, we summarize the findings of the two workshops, define a set of dimensions to characterize various software architecture evaluation techniques, and make concrete recommendations for implementing architecture evaluation practices.

1 Introduction

Because of the importance of architectural decisions, it is fitting that they receive close scrutiny. Few will argue against the cost-effectiveness of quality evaluation as early as possible in the software development life cycle. If problems are found early in the life cycle, they are trivial to correct—a change to a requirement, or specification, or design is all that is necessary. Software quality cannot be appended late; it must be inherent from the beginning, built-in by design. The definition of a software architecture is the earliest point in the life cycle of a system where an artifact exists that can be checked to determine how well the system will meet its requirements—both explicit and implicit. It is in an organization's best interest to evaluate (and reject, if necessary) prospective design candidates during the design phase, before long-term institutionalization occurs. Therefore, an effective technique to evaluate a candidate architecture—*before* it becomes the accepted blueprint—is of great economic value.

Furthermore, when *acquiring* a large software system that will have a long lifetime within the acquiring organization, it is important that the organization develop an understanding of the underlying architecture of candidate systems. This understanding allows an assessment of the suitability of the candidates with respect to qualities of importance, particularly with respect to how the candidates will support the future needs of the organization.

On the other hand, it should be noted that an architecture cannot guarantee the functionality or quality required of a system. Poor downstream design, implementation, testing, or management decisions can always undermine an acceptable architectural framework. Decisions at all stages of the life cycle—from high-level design to coding and maintenance—affect quality. A

software architecture evaluation assesses the ability of the architecture to support the desired qualities. Refinements of the architecture into implementation that preserve the qualities are necessary for the final product to actually achieve those qualities.

Our primary recommendations are that an architecture evaluation be held as early as possible in the life cycle of the project, and that the evaluation be conducted as a formal review. Ideally, such a review is held early enough so that decisions that have been made can be revoked without extensive cost, but late enough so that bindings to hardware have been made so that performance characteristics of the system can be analyzed. This places the evaluation somewhere in the early phases of high-level system design.

There are a number of generic recommendations that exist for carrying out any formal review, such as assembling the correct people, making wise use of their time, and so forth. Since our goal is to provide a full description of an architecture evaluation, we will no doubt include without distinction many activities and recommendations that make sense for any review.

Evaluation connotes measuring value using a quantitative scale. Given our current understanding, quantitative practices are rare at the architectural level, so the term evaluation is premature. We will use the word evaluation to mean a process by which we draw conclusions about an architecture. A review refers to a gathering of individuals to conduct an evaluation. Our long-term goal is software architecture evaluation in the true sense of the word.

1.1 SEI Workshops

All of the evidence in this report comes from a series of two workshops on industrial practice of software architecture evaluation held over the past year at the SEI. The participants were invited based upon our knowledge of their company's experience with and/or commitment to architecture evaluation. Participants gave presentations on their industrial experience and provided reference materials to help us understand their techniques. Those with the most experience also completed a detailed questionnaire (included in the appendix) on specific aspects of their industrial experience. Together we discussed the issues and consolidated our experience to form the backbone of this report.

The participants fell into three categories:

- industrial participants in charge of defining and conducting architecture evaluations within their company (we refer to these as "internal")
- consultants from industry and academia who, as outside experts, conduct evaluations for other companies
- academic/industrial observers, some having evaluation experience

The first set of practitioners met at the SEI on November 9-10, 1995. The second set met at the SEI on May 9-10, 1996. The total set of participants included:

Gregory Abowd, Georgia Institute of Technology

Len Bass, SEI

Joe Batman, SEI

Paul Clements, SEI

Ron Crocker, Motorola

Stu Feldman, IBM T.J. Watson Research

David Garlan, Carnegie Mellon University

Christine Hofmeister, Siemens

Kalai Kalaichelvan, Nortel

Rick Kazman, SEI

Philippe Kruchten, Rational

Joe Maranzano, AT&T Bell Laboratories

Linda Northrop, SEI

Connie Smith, Performance Engineering Services

David Weisman, Lockheed Martin

The domains of systems reviewed by the participants include: transaction processing/customer care, telecommunications, air traffic control, decision support systems, event management systems, revision control systems, information management, and others.

1.2 About This Document

This report describes a recommended best practice for performing software architecture evaluation. It is a consequence of two workshops held at the SEI, the first on November 9-10, 1995 and the second on May 9-10, 1996. The report is organized into five major sections: Introduction, Costs and Benefits, Categorization of Evaluation Techniques, the Recommended Best Practice, and Recommendation Summary. The two sections following this introduction lay out first some economic and then some technical groundwork essential to understanding the recommendations that follow.

In Section 4, the recommended best architecture evaluation practice is described in terms of the preconditions to, the activities during, and the output of the evaluation. Section 5, Recommendation Summary, recaps the major recommendations and discusses the rationale behind them. The appendix contains the workshop questionnaire.

2 Costs and Benefits

There are a number of motivations for performing an architecture evaluation, but there is an associated cost in terms of time and committed resources. An organizational issue is to decide whether the benefits are worth the cost. We start, therefore, by reporting what is known about costs and then give the perceived benefits of an architecture evaluation.

2.1 Costs

Some of the workshop participants maintain records to help determine the cost of a full architecture evaluation. For example, AT&T, having performed approximately 300 full-scale architecture evaluations on projects requiring a minimum of 700 staff days, reported that the average cost of an architecture evaluation review was, based upon estimates from individual project managers, 70 staff days. In addition, two of the consultants who performed architecture evaluations reported cost estimates. Rational Software Corporation has performed around 30 evaluations and charges an average cost of $50K for projects of at least 500 KLOC in size. The SEI Software Architecture Analysis Method (SAAM) evaluators have performed 10 evaluations for projects ranging in size from 5-100 KLOC and report a cost of 14 days. Most participants also noted that there are increased start-up costs for an organization beginning an architecture review practice due to a lack of architectural maturity in the company.

Most of the organizations represented (AT&T, Siemens, Nortel, Motorola, and Lockheed Martin) have established corporate units responsible for defining and conducting architecture evaluations. One of the main reasons for this centralization is that each company is interested in maximizing the amount of corporate reuse at the architectural level. All of these companies reported that the individual start-up costs for such an organization were non-trivial.

We recommend that architecture evaluation teams not include members of the development project. Instead, individuals are selected from within their company to serve on architecture evaluation teams based on their past performance and demonstrated skills with large-scale system organization and design. This membership issue surfaces two cost-related concerns. First, there is the worry of reduced organization-wide development productivity because superior designers are removed from active involvement. This cost can be mitigated by making membership on the architecture evaluation board non-permanent. The second concern stems from this temporary engagement on the architectural evaluation team. Each new evaluator needs to be trained in the evaluation techniques and gain experience in evaluation before being a productive member of the evaluation team. There is a cost associated with this learning curve for architecture evaluators.

2.2 Benefits

The benefits of an architectural evaluation can be categorized as follows.

2.2.1 Financial

At AT&T, each project manager reports perceived savings from an architecture evaluation. On average, over the past eight years, projects receiving a full architecture evaluation have reported a 10% reduction in project costs. Given the cost estimate of 70 staff days, this reported 10% cost reduction illustrates that on projects of 700 staff days or longer, the evaluation pays for itself.

Other workshop attendees did not have strongly quantified data, but several consultants reported that over 80% of their business was repeat business. Their customers recognized sufficient value in architecture evaluation to be willing to pay for additional evaluations.

Workshop participants also reported several anecdotes of estimated cost savings for the customers of their evaluations. One consultant reported that, as a result of an architecture evaluation, a large company avoided a multimillion dollar purchase when the architecture of the global information system they were procuring was, upon evaluation, not capable of providing the desired system attributes necessary to support a product line. Another reported that early analysis of an electronic funds transfer system showed a fifty billion dollar transfer capability per night: half the desired capacity. An architecture evaluation of a retail merchandise system revealed early that there would be peak order performance problems that no amount of hardware could fix; a major business failure was prevented. In the architecture evaluation of a revision control system, a number of severe limitations in achieving system portability and modifiability were uncovered. A major redesign of the system was recommended.

It is also useful to point out cases where architecture reviews did not occur. One participant described how a rewrite of a customer accounting system was estimated to take two years. After seven years, the system had been re-implemented three times. Performance goals have never been met, despite the fact that the latest version uses sixty times the CPU power of the original prototype version. In a large engineering relational database system, performance problems that were largely attributable to design decisions made integration testing impossible. The project was cancelled after twenty million dollars had been spent.

2.2.2 Increased Understanding and Documentation of the System

One of the benefits of any formal review is that it forces the reviewee to prepare for the review. By giving the reviewees an indication of the focus of the architecture evaluation and the requirement for a representation of the architecture before the review is held, the reviewees are required to document the system's architecture. Many systems do not have a top-level architecture that is understandable to all of the developers. The architecture is either too brief or, more commonly, too long, perhaps thousands of pages. Furthermore, there are often misun-

derstandings among developers about some of the assumptions for their components. The process of preparing for the review will reveal many of these problems.

Furthermore, the architecture evaluation focuses on a few specific areas with specific questions to be answered. Answering these questions usually involves giving an explanation of the design choices and their rationales. Having a documented design rationale is important later in the life cycle so that the implications of modifications can be assessed. Capturing design rationales after the fact is one of the more difficult tasks in software development. By capturing the design rationale as presented in the architecture review (even by low-cost methods such as videotaping), invaluable information is available for later use.

2.2.3 Detection of Problems with the Existing Architecture

The third benefit of an architecture evaluation is early detection of problems with the existing architecture. The earlier in the life cycle that problems are detected, the cheaper it is to fix them. The problems that can be found by an architectural-level inspection include unreasonable (or expensive to meet) requirements, performance problems, and problems associated with potential downstream modifications. For example, an architecture evaluation which exercises scenarios of typical system activity can manifest rough performance specifications. Exercising system modification scenarios can reveal portability and extensibility problems that will be especially critical if the architecture is to support a product line, rather than just a single product. Architecture evaluation, therefore, provides early insight into product capabilities and limitations.

2.2.4 Clarification and Prioritization of Requirements

The fourth significant benefit is validation of requirements. Discussion and examination of how well an architecture meets requirements also opens requirements up for discussion. What results is a much clearer understanding of the requirements and, usually, a prioritization of the requirements. Requirements creation, when performed in isolation from a high-level design, usually results in specification of conflicting system properties. High performance, security, fault-tolerance, and low cost are all easy to demand but difficult to achieve, and often impossible to achieve simultaneously. Architecture evaluations uncover the conflicts and tradeoffs, and provide a forum for their *negotiated* resolution.

One workshop participant reported reviewing a system whose performance requirements (a seemingly innocuous three-minute update rate) would have necessitated the use of twenty large-scale high-cost CPUs. The “requirement” turned out to have been ill-conceived; relaxing it to four minutes produced an acceptable system at an acceptable cost.

2.2.5 Organizational Learning

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

In summary, architecture evaluations tend to increase quality, control cost, and decrease budget risk. Architecture is the framework for all technical decisions and, as such, has a tremendous impact on cost and quality of the product. An architecture review does not guarantee high quality or low cost, but it points out areas of risk in a design. Other factors such as testing or quality of documentation and coding will contribute to the eventual cost and quality of the system.

Benefits other than the immediate ones of helping to facilitate the construction of the system also accrue. Architectures are manifested in organizational entities (for example, work breakdown structures are based on system decompositions of functionality) but they also reflect existing organizational structures. If, for example, portions of a system are being developed by subcontractors, then recognizing commonalities across the portions of the system being developed by the two different subcontractors is difficult. An architecture evaluation that has maintenance as one of its focus areas will detect commonalities regardless of organizational considerations.

3 Categorization of Evaluation Techniques

Workshop participants described a variety of techniques used to perform an architecture evaluation. Each of the techniques has a different cost and can be used to elicit different information. We found that there are two basic categories of these techniques: those that generate qualitative questions to ask of an architecture and those that suggest quantitative measurements to be made on an architecture. We found that questioning techniques can be applied to evaluate an architecture for any given quality. In fact, it is consideration of the quality that drives the development of questions (in the form of checklists or scenarios, as described below). Questioning techniques, however, do not directly provide a means for answering those questions. Measuring techniques, on the other hand, are used to answer specific questions. In that sense, they address specific software qualities (for example, performance or scalability) but are not as broadly applicable as questioning techniques.

Each of these techniques will be described briefly below, as background for its recommended application.

3.1 Questioning Techniques

We identified three questioning techniques: scenarios, questionnaires, and checklists. These techniques differ from each other in applicability, but they are all used to elicit discussion about the architecture and increase understanding of the architecture's fitness with respect to its requirements.

3.1.1 Scenario

People often want to analyze software architectures with respect to quality attributes expressed using words like *maintainability, security, performance, reliability*, and so forth. These words provide convenient ways for describing and communicating a host of common, recurring problems in software. However, most software quality attributes are too complex and amorphous to be evaluated on a simple scale, in spite of our persistence in describing them that way. Furthermore, quality attributes do not exist in isolation, but rather only have meaning within a context. A system is modifiable (or not) with respect to certain classes of changes, secure (or not) with respect to specific threats, usable (or not) with respect to specific user classes, efficient (or not) with respect to its utilization of specific resources, and so forth. This notion of context-based evaluation of quality attributes has led to the adoption of scenarios as the descriptive means of specifying and evaluating quality attributes *within a context*. A scenario is a specified sequence of steps involving the use or modification of the system. Scenarios provide a means to characterize how well a particular architecture responds to the demands placed on it by those scenarios. Scenarios test what we normally call modifiability (by proposing a set of specific changes to be made to the system), security (by proposing a specific set of threat actions), performance (by proposing a specific set of usage profiles that tax resources), and so on.

As an aid to creating and organizing scenarios, we appeal to the concept of *stakeholders* related to the system. Relevant stakeholders include: the person responsible for upgrading the software—the end user; the person responsible for managing the data repositories used by the system—the system administrator; the person responsible for modifying the runtime functions of the system—the developer; the person responsible for approving new requirements for the system, etc. The concept of stakeholders matches the difference between runtime qualities and developmental qualities, that is, those qualities that are a function of the system's execution (such as performance), and those that reflect operations performed offline in a development environment (such as modifiability). Scenarios of the former variety will eventually be performed by stakeholders such as end user—the latter by developers or maintainers.

3.1.2 Questionnaire

A questionnaire is a list of general and relatively open questions that apply to all architectures. Some questions might apply to the way the architecture was generated and documented (by asking if there is a designated project architect or if a standard description language was used). Other questions focus on the details of the architecture description itself (by asking if all user interface aspects of the system are separated from functional aspects). The evaluation team is looking for a favorable response and will probe a single question to a level of detail that is necessary to satisfy their concern. An example of a questionnaire-based evaluation process (that happens to go beyond architectural investigation) is the software risk evaluation process [Joseph 94] developed by the Software Engineering Institute. The utility of a questionnaire is related to the ease with which the domain of interest can be characterized and circumscribed.

3.1.3 Checklist

"Just as a building inspector uses a checklist to ensure that a structure is in compliance with the building codes, software architects use a checklist to help them keep a balanced focus on all areas of the system." [AT&T 93] A checklist is a more detailed set of questions that is developed after much experience evaluating a common (usually domain-specific) set of systems. Checklists tend to be much more focussed on particular qualities of the system than questionnaires are. For example, performance questions in a real-time information system will ask whether the system is writing the same data multiple times to disk or whether consideration has been given to handling peak as well as average loads.

3.1.4 Summary

There is a natural relationship between these three questioning techniques. Scenarios are intended to be system-specific questions. Experience evaluating a family of related systems can result in generalizing a set of commonly used scenarios, turning them into either domain-specific entries in a checklist or more general items in a questionnaire. Checklists and questionnaires reflect more mature evaluation practices. Scenarios can reflect less mature evaluation

practices. Another difference is that scenarios, since they are system specific, and are developed as part of the evaluation process. Checklists and questionnaires are assumed to exist before a project begins.

3.2 Measuring Techniques

Measuring techniques result in quantitative results. Rather than provide ways to generate the questions that will be asked about an architecture, these techniques provide answers to questions an evaluation team might already have about particular qualities of an architecture. Since the questions almost always precede the answers, we can see these measuring techniques as more mature than the questioning techniques. In fact, we only saw evidence of measuring techniques being used to answer questions of performance or modifiability.

3.2.1 Metrics

Metrics are quantitative interpretations placed on particular observable measurements on the architecture, such as fan in/fan out of components. The most well-researched measuring techniques provide answers on overall complexity that can suggest places of likely change (as shown by Selby & Reimer [Selby 95]). With measuring techniques, the evaluation needs to focus not only on the results of the measurement/metric, but also on the assumptions under which the technique was used. For example, a calculation of performance characteristics makes assumptions about patterns of resource utilization. How valid are these assumptions? Coupling and cohesion metrics make assumptions about the types of functionalities embodied in the components being examined. How valid are these assumptions?

3.2.2 Simulations, Prototypes, and Experiments

Building a prototype or a simulation of the system may help to create and to clarify the architecture. A prototype whose components consist of functionless stubs is a model of the architecture. Performance models are an example of a simulation. The creation of a detailed simulation or prototype just for review purposes is typically expensive. On the other hand, these artifacts often exist as a portion of the normal development process. In this case, using these artifacts during a review or to answer questions that come up during the review becomes a normal and natural procedure.

An existing simulation or prototype may be an answer to an issue raised by a questioning technique. For example, if the evaluation team asks “What evidence do you have to support this assertion?,” one valid answer would be the results of a simulation.

3.3 Comparison of Categories of Evaluation Techniques

We can further distinguish architecture evaluation techniques across a number of different dimensions. Table 3-1 shows a summary of the classification of these evaluation techniques.

Review Method	Generality	Level of Detail	Phase	What is Evaluated
Questionnaire	general	coarse	early	artifact process
Checklist	domain-specific	varies	middle	artifact process
Scenarios	system-specific	medium	middle	artifact
Metrics	general or domain-specific	fine	middle	artifact
Prototype, Simulation, Experiment	domain-specific	varies	early	artifact

Table 3-1: Properties of the Evaluation Approaches

The dimensions are described below.

3.3.1 Generality

General-purpose techniques focus on general-purpose issues and can be applied to any architecture. Domain-specific techniques focus on issues particular to a given domain. As we already discussed, questionnaires are a general-purpose approach, whereas checklists are domain specific. Scenarios, at least as they are initially defined, tend to be system specific but may be either, depending on the particular issue a scenario is addressing. Metrics are mostly general purpose, although there may be some domain-specific metrics, for example in the telecommunications domain. Prototypes, simulations, and experimentation are primarily domain- specific, although there are some general-purpose simulation generation tools.

3.3.2 Level of Detail

Level of detail indicates how much information about the architecture is required to perform the evaluation. Values here represent a continuum and usually determine when in the development cycle the evaluation technique can be applied. Coarse-grained approaches can be applied early in the design process for they do not require much detailed information such as component specification or connection protocols. Fine-grained approaches generally need more detail and hence must be applied later, when more decisions have been bound.

3.3.3 Phase

There are three phases of interest to architecture evaluation: early, middle, and post-deployment. The entry in the table lists the *earliest* that a particular approach can be employed. In general, the approach should be applied as early as possible.

Early-phase evaluation occurs after initial high-level and high-priority architectural decisions have been made. At this point, we can evaluate the preliminary decisions and detect poor preliminary choices. An actual architecture does not yet exist, just the preliminary decisions do.

Middle-phase evaluation occurs after some elaboration of the architectural design. Elaboration is an iterative process; the evaluation can occur at any point here. At this point, there should be an actual architectural design in varying stages of completeness (depending on how much has been elaborated). At this stage we can identify problems with the architectural design.

Post-deployment phase evaluation occurs after the system has been completely designed, implemented, and deployed. At this stage, both the architecture and the system exist, so we can answer additional questions about whether the architecture matches the implementation. If the product has been in existence for a while, we can also check for architectural *drift*—movement away from the original design. These are different kinds of issues than the ones we discuss in this report.

3.3.4 What Is Evaluated

There are two different kinds of questions that could be answered in an evaluation. The first has to do with the architecture as an *artifact* or product. Here the focus is on evaluating the architecture—properties that should hold, or issues to look at, such as minimal connection between components or encapsulation. This is the most common subject of an evaluation. Another kind of evaluation looks at the *role* played by the architecture in the development process. This explores issues of how the architecture is used in the product's life cycle, whether it is useful, and who has responsibility for it.

Questionnaires and checklists can evaluate both the artifact and the role, depending on what kind of question is asked. Examples of questions that address the process at the early phase include "Do you have an architect?" and "How will you go from here to develop the architecture?" Examples of questions that address the process at a later stage include "Do people understand the architecture?" and "What happens if we split the development team?" Examples of questions at the post-deployment stage include "Is the architecture adhered to when you change the system?" and "Does the implementation match the architecture?"

Scenarios, metrics, and prototypes by their nature are geared toward evaluating the artifact only. It might be possible to come up with scenarios that focus on the role, but that is not really their intent.

3.4 What Technique to Use

The recommended technique depends on the development process, the maturity of architecture evaluation within an organization, and the particular qualities being examined during the evaluation.

If during the development process, simulations, prototypes, or experiments have been developed, they are the recommended techniques to use to provide information during an architectural review that is within their scope. That is, a prototype may have been developed to test particular performance characteristics. The use of this prototype to answer questions concerned with modifiability is likely inappropriate. Specifically, simulations and prototypes are recommended to answer performance questions.

Questionnaires and checklists are evolved over time and so if an organization is just beginning to perform architecture evaluations and does not have questionnaires and checklists in place, scenarios are the technique of choice. Development of a collection of scenarios is an activity performed during an evaluation, and so, after performing a collection of evaluations, an organization can build a database of scenarios and turn them into questionnaires and checklists. Even if an organization has questionnaires and checklists already in place, scenarios are recommended to deal with issues not covered in the questionnaires or checklists, and can also be used to grow the questionnaires and checklists for future evaluations.

That is, scenarios are recommended as a method of getting started with a new area of evaluation, and later can be grown into questionnaires and checklists.

4 The Recommended Best Practice

We have summarized the costs and benefits for architecture evaluation and categorized five different evaluation techniques. Next, we recommend the best practices for preparing, executing, and reporting on an architecture evaluation.

4.1 Evaluation Preconditions

Preconditions are the set of necessary assets and conditions that must be in place before a successful evaluation can proceed. Preconditions include an understanding of the evaluation context, involvement of the right people, organizational expectations and support, evaluation preparation, and an appropriate representation of the architecture being examined.

4.1.1 Understanding of the Evaluation Context

Planned or Unplanned

Architecture-level project evaluations usually occur in one of two modes. In the first, the evaluation is considered a normal part of the project's development cycle. The review is scheduled well in advance, built in to the project's work plans and budget, and follow-up is expected. In the second mode, the evaluation is unexpected and usually the result of a project in serious trouble and taking extreme measures to try to salvage previous effort. These two kinds of evaluation are fundamentally different; they have different goals, different agendas, are subject to different expectations, and produce different results.

The planned evaluation is ideally considered an asset to the project, at worst a good chance for mid-course correction. The review can be perceived not as a challenge to the technical authority of the project's members, but as a validation of the project's initial direction. Planned evaluations are proactive.

The unplanned evaluation, however, is more of an ordeal for project members, consuming the project resources and schedule from a project already struggling with both. It tends to be more adversarial than constructive. It is used only when management perceives that a project has a substantial possibility of failure and needs to make a mid-course correction. Unplanned evaluations are reactive.

Because this document is a recommendation of best practices, we will not deal with unplanned reviews. It is the recommendation of this report that architecture evaluations be an integral part of the development process, planned and scheduled in advance, along with follow-up activities. When architecture evaluations are performed as a planned part of the software life cycle, unplanned evaluations should never be required.

Discovery or Validation

Some practitioners recommend an early, lightweight “architecture discovery review,” at a point during development after requirements have been set but before architectural decisions have become firm. In this context, the requirements have not yet been validated in the context of a formal mapping to an architecture or design approach. This is the ideal time to challenge requirements on the basis of their feasibility or cost of implementation.

Such a discovery review would be held in addition to (not in lieu of) the type of full architecture evaluation that examines a set of architectural decisions against a presumably unbending set of functional and quality requirements.

These two reviews attack different kinds of decisions and can analyze for different kinds of qualities. The discovery review uncovers a delicate balance that must be respected. If no architectural decisions have yet been bound, there can be nothing to discuss. If decisions have been strongly bound by this stage, then some expense will be incurred by changing them. The prime assumption of discovery reviews is that some decisions have been made, but that these decisions are typically *not* strongly bound to an architecture and can be changed without great expense. Discovery reviews are less costly, but since most architectural decisions have not been evaluation-determined at the time of this review, the perceived benefits are less.

Purpose of the Evaluation

It is important that the scope of the evaluation be kept under control. An evaluation could investigate many things, including whether the project’s intended functionality can be achieved and whether all quality goals can be met. In order to focus the evaluation, we recommend that a small number of explicit goals be enumerated. The number of goals upon which to focus should be kept to a minimum, around three to five. An inability to define a *small* number of high-priority goals for the evaluation is an indication that the expectations for the review (and perhaps the system) may be unrealistic. These goals define the purpose of the evaluation and should be made an explicit portion of the evaluation contract discussed in Section 4.1.3.

Size of System Being Evaluated

There is a cost associated with any review, and the benefits should exceed the cost. The type of evaluation we are advocating here is suitablfor medium- and large-scale projects but may not be cost-effective for small projects. Based on the costs of the evaluation discussed in Section 2, we recommend an evaluation for projects that consume more than four person-years of effort.

4.1.2 The Right People

There are two groups of people whose participation is essential to a successful evaluation. Representatives from each of these groups must be available at some point during the architecture evaluation.

Project Representatives

The project should be represented by the system architect, the designers for each major component, and the system stakeholders.

It is imperative to secure the time of the project's architect, if available, or at least someone who can speak authoritatively about the architecture and design. Although motivation for design decisions is important, this person (or these people) should be primarily able to communicate the facts of the architecture quickly and clearly.

For very large systems, the designers for each major component need to be involved to ensure that the architect's notion of the system design is in fact reflected and manifested in the lower levels of the design.

The project's stakeholders must be identified, so that their interests will be represented in the evaluation. Stakeholders include developers, managers, customers, different types of users, installers, maintainers, contract monitors, and subcontractors. If the system must interact with other systems, the set of stakeholders must include representatives of the other systems as well—possibly even including the organization's competitors. Individuals should be assigned who can speak for these roles. It is essential to identify the customers for the evaluation report and to elicit their values and expectations.

Evaluation Team

As we have already indicated, software architecture evaluation teams ideally are separate entities within a corporation. The evaluation team must be assembled in a way that addresses the following:

- The team must be perceived by the development project as impartial, objective, and respected. The team must be seen as being composed of people appropriate to carry out the evaluation, so that the project personnel will not regard the evaluation as a waste of time, and so that the team's conclusions will carry weight.
- The team members should expect to spend at least 100% of their nominal work time on the evaluation as it runs its course. Late-night planning sessions and off-the-record breakfast meetings with project personnel are the norm.
- The team should include people highly fluent in architecture and architectural issues and be led by someone with solid experience in designing and evaluating projects at the architectural level.
- The team should include at least one system domain expert—someone who has built systems *in the area being evaluated*.
- The team should include someone to handle logistics: scheduling meetings, reserving conference rooms, handling travel arrangements, making copies of documentation, distributing read-ahead material, ordering lunches, procuring supplies, and so forth.

The team should include a librarian responsible for organizing and making available the documentation about the project.

- The team should be located as close to the source of the artifacts it will be examining as possible. Locality will simplify logistics and enhance communication with project personnel.
- The team, ideally, should include an “apprentice” reviewer to propagate architecture review capability, as well as provide for reviewer turnover to avoid burnout.

In addition, the team must have access to the following resources:

- Applicable domain knowledge is crucial and must be available from consultants to the team, if not resident on the team itself.
- Access to the design documents, any working prototypes, or even (on rare occasions) the source code, if it exists.
- Knowledge of the evaluation criteria, such as might be represented by a performance engineer, if not resident on the team itself.
- Support staff will be needed to assist in review process logistics and report generation.

4.1.3 Organizational Expectations and Support

Critical to evaluation success is a clear, mutual understanding of the expectations of the organization sponsoring the review, as well as support for the review in terms of organizational resources.

Contract

Senior managers need to set expectations for the review to both the evaluation team and the project personnel. A mechanism for this is a contract that determines

- who will be told what upon completion of the evaluation
- what will be the evaluation criteria (and, by implication, what will not)
- what and who will be made available to the evaluation team
- what follow-up is expected on the part of the evaluation team and the project
- expectations (for the project) of the maximum time the review is expected to take

The process of negotiating the contract may be as important as the contract itself. The eye-to-eye communication necessary to forge a contract may be the largest factor in setting realistic expectations.

It should be understood that analysis at the architectural level is not a definitive analysis. All that can be determined at the architectural level is that it is feasible that the resulting system will have the desired qualities. There are no guarantees: a poor implementation of a good design will likely perform poorly. Thus, an important part of expectation setting is making sure that all review stakeholders understand the nature of the results.

Supporting Culture

A clear understanding of the supporting culture within the organization is essential. How ingrained does the organization make architecture evaluations? In particular, the following questions should be answered:

- Is the evaluation part of the organization's standard project life cycle?
- Is it considered career-enhancing to serve on an evaluation team?
- Does the organization recognize a standing capability to perform an architecture evaluation as a core competency that is worth investing in? Is the organization willing to invest the necessary resources such as the long-term diversion of one or more valuable architects from development projects to cultivate evaluation capability?
- Is there a standing evaluation organization? Although evaluators may rotate on and off the evaluation team, there are tangible benefits to keeping a core evaluation team. They include
  - Working relationships are established and maintained.
  - Experience can be transferred from one evaluation to the next, thus building up a corporate knowledge base about techniques, criteria, methods, and approaches.
  - The normal inertial start-up time of a team with members who are unfamiliar with each other can be avoided.

4.1.4 Evaluation Preparation

Certain materials must be prepared to permit a successful architecture evaluation.

Agenda or Goals

An agenda is needed for focus. The agenda should be detailed, but flexible. Often, information elicited during a review will launch an area of investigation not previously considered. If the architecture evaluation does not take the form of a review meeting (for example, the evaluation could be a report which an independent team prepares), the goals for the evaluation, as well as the evaluation criteria, must be spelled out clearly. For example, rather than saying that "our system should be integrable," a criterion should state a more specific goal, such as "our system should integrate with system X with no more than 10 person-days of effort."

Read-Ahead Materials for Project Personnel

Read-ahead material that offers some guidance for review preparation should be provided by the evaluation team to project personnel. Ideally, this is a checklist of questions that the reviewers will ask.

If questionnaires or checklists are going to be used during the evaluation, they should be presented to the development group prior to the review. In fact, if these are organization-wide artifacts, they should be presented to the development group when the project is initiated.

Ensuring that the development group has the questions prior to the review will enable them to prepare answers and make adequate preparations; otherwise, review time must be spent preparing the responses. It may also have the long-term effect that the development groups will steer their designs to perform well on the evaluation—which, as long as the evaluation adequately addresses the issues important to the organization, is the whole point.

Scenarios are system-specific and should be created to reflect all of the stakeholders' concerns. This requires input from the various stakeholders. Prior to the review, the evaluation team should iterate with the various stakeholders and develop a collection of scenarios on which the evaluation will be based.

Read-Ahead Materials for Evaluation Team

The evaluation team needs access to materials that describe the architecture and discuss the rationale behind architectural decisions. Expectations should be set about the level of detail of the material: 50 pages would be preferable to 500 or 1. Ideally this material will address the preestablished checklist of questions that the reviewers will ask. This is important, because it not only prepares the development team to answer these questions clearly, but it can motivate them to address these issues in the architecture. That is, the knowledge that such questions will be asked during an evaluation can positively affect the architecture, by forcing it to explicitly address these questions from the earliest stages of design.

This assumes that the architecture and the requirements are in a concrete enough form to be unambiguously documented and that they have been. If this is not the case, elicitation of this information will be the first order of business in the evaluation itself.

In organizations in which this evaluation process is part of the culture, it is far less likely to be the case that the architecture and requirements have not been adequately documented by the time the evaluation begins. In organizations without standard representations for requirements and architectures, the evaluation team may distribute templates for desired documents that illustrate to project personnel the kind of information required and approved forms for conveying it.

Ranked Requirements

We have already stressed the importance of scoping the evaluation. One concrete way to do this is to rank quality and function requirements of the system before the evaluation begins. Requirements frequently are in conflict and this prioritization will help to resolve those conflicts when they arise. Industrial experience suggests that three to five is a reasonable maximum number of quality attributes to try to accommodate. Such a ranking of requirements serves the project by preventing the architecture from being muddied by the lobbying from representatives of a large number of conflicting quality attributes. If more desired quality attributes are expressed, they should be ranked as "above-the-line" or "below-the-line." Attributes that are

"above-the-line" *must* be addressed by the architecture. The other attributes, while desirable, may be sacrificed due to deadlines, budgets, or increased software complexity.

4.1.5 Architecture Representation

We have been assuming up to this point that some description of the architecture is available before the evaluation. Over time, the culture of an organization that favors architecture will ensure this. But initially, it will likely be the case that no such documentation exists. In such cases, the elicitation of the architectural description becomes a part of the evaluation process. Most consultants assume this mode of operation.

It was clear from our workshops that no standard architecture representations (beyond box-and-line diagrams) have been used in formal architecture evaluations. Nor is it likely that such a representation language will emerge very soon. Most of our evidence for this comes from AT&T where they have been doing these evaluations for eight years without a standard description language. Though Rational Software Corporation supports several object-oriented architecture representations, the architectural descriptions are done by external consultants and not the project teams [Kruchten 95].

Whereas no formal architectural representation language has emerged, we did notice two kinds of architectural descriptions that are used: criteria-driven and representation-driven, described below.

Criteria-Driven Architecture

For some reviews, the representation of the architecture is not a critical issue as long as the architecture's "fitness to purpose" can be determined. In this case, the architectural information provided for the evaluation is dependent upon the specific review criteria, often expressed by a set of scenarios or the high priority questions in a questionnaire or checklist. Declaring a specific quality as important (for example, security) will result in certain aspects of the system being revealed in the architecture and others hidden. A specific set of scenarios will determine, for instance, the granularity at which a set of architectural components and interconnections should be expressed. Change the scenarios and the architecture description is also likely to change.

Representation-Driven Architecture

If there is a standard representation language, such as Krutchen's "4+1" approach [Kruchten 95], or Software Architecture Analysis Method's simplistic data/control flow notation [Kazman 96], the information provided by the architecture description is dictated by the notation. Determining whether the description is complete is usually left to the evaluation team. Over time however, questions asked of the architecture will tend to be ones that can be answered by the representation. An organization needs to guard against having the representation drive the evaluation criteria.

4.2 Evaluation Activities

During an evaluation, there are activities to evaluate the system and activities to record and prioritize the results. The execution of these activities will depend upon the precise evaluation technique employed (see Section 3).

4.2.1 Recording And Prioritizing

During the evaluation, the evaluation team is attempting to highlight critical issues that are raised in support of or (more importantly) against the architecture. Each issue raised during the review is documented. For example, these could be written on index cards and pinned to a board, or stored in a widely accessible location, such as a corporate database. At convenient points during the evaluation, the participants' comments on critical issues are organized by theme. The evaluation team then rates the themes as *project-threatening*, *major*, or *minor*. Thus, at the conclusion of the review, there is a prioritized list of issues that can be reported to higher management and used to determine follow-up activities.

4.2.2 Evaluating

There are many qualities that can be the subject of an evaluation. As one of the preconditions of the evaluation, the primary qualities to examine have been agreed upon. The most common ones to come up in the workshop were cost, functionality, performance, and modifiability. We address these in turn below.

For Cost

Every architecture evaluation should consider the cost consequences of the proposed design. There are a variety of cost models that can be used to estimate cost and most are organization-dependent. The goal is not to come to a precise determination of projected cost during an architectural evaluation, but to see if there are any features in the architecture that are suspected to be sources of undue risk and, potentially, unbound cost.

For Functionality

Sometimes it is unclear where in the architecture particular functionality is to be executed or how the various aspects of the functionality are distributed across the components of the architecture. The evaluation team should understand the essential functions of the system and should look to see how that function is unambiguously defined in the architecture representation or described succinctly by the architect. A response to the question, "How is X done?" that takes the form, "X could be done either here or there," indicates a potential problem that should be recorded as an issue.

Distributing functionality across several components is not an issue of concern for functionality, although it could be an issue for modifiability. Other functionality-related issues that are important to examine include

- making sure there are written requirements in all key areas
- checking that the functionality of the architecture system is complete with respect to these functional requirements
- checking that the system acceptance criteria exist

For Performance

To evaluate performance, usually in terms of resource utilization, several pieces of information are required [Smith 94]. These include

- work load information, which for the scenario under investigation consists of the number of concurrent users, request arrival rates, and performance goals
- a software specification in performance terms, which consists of execution paths and programs, components to be executed, the probability of execution of each component, the number of repetitions of each software component, and the protocol for contention resolution used by the software component
- environmental information, where binding decisions have been made. This information consists of system characteristics such as configuration and device service rates, overhead information for "lower level" services, and scheduling policies

Resource utilization can be derived from the three types of information given above; it consists of CPU usage (or number of instructions executed per unit time), input/output activity, number of database calls, amount of communication and other device usage, amount of service routine usage, and memory usage. This resource usage is then compared to the resource budget for the system to determine (as a rough estimate) whether a performance issue exists.

For Modifiability

Modifiability questions are best addressed via scenario-based techniques. Modifiability evaluation depends both on individual scenario evaluations and on the cumulative effect of all of the scenarios.

Effects of Individual Modifications

Each modification scenario is reviewed from the point of view of the modification's difficulty. This difficulty depends on how many different components need to be modified and how difficult the modification of each component will be. The technique to determine the difficulty is similar to the technique used for tracking requirements. That is, the developers explain how the modification is to be performed and what the difficulties will be. Indications of problem areas should be recorded.

Effects of Overlapping Modifications

Modifications to an architecture may interact with each other. That is, if one component must be changed to support two unrelated modifications, those modifications are more likely to be a source of problems (that is, a source of increased complexity) than two similar modifications performed on distinct components. Thus, the components affected by multiple modifications need to be examined in an attempt to estimate the cumulative complexity of these changes. One simple estimation technique for this is to add together the difficulty of each individual modification. This gives a first-order estimate in line with the accuracy of architectural estimates in general.

4.2.3 Reviewing Requirements

Each serious issue that is raised during the evaluation needs to be examined both from the point of view of modifying the architecture and from the point of view of modifying the requirement(s) that caused the issue to be raised. For example, if there is a performance issue, it can be resolved by either increasing the performance of the system in some manner or reexamining the performance requirement. Sometimes requirements are set without careful examination of the system context, and the specific requirement that is driving an issue should be revisited. There also may be a tendency to "gold-plate" a system by requiring functionality and performance levels that aren't really needed. Sometimes this is justified by a business case but other times this should be resisted. An architecture evaluation is an early opportunity to ferret out nonessential or conflicting requirements.

4.2.4 Reviewing Issues

All of the issues that were raised during the review, as well as their ongoing ranking, should be discussed in feedback made available to the development unit. In addition, there should be a check for the following early warning signs:

- The architecture is forced to match the current organization.
- Top-level architecture components number more than 25.
- One requirement drives the rest of the design.
- The architecture depends upon alternatives in the operating system.
- Proprietary components are being used when standard components would do.
- The component definition comes from the hardware division.
- There is too much complexity (e.g., two databases, two start-up routines, two error-locking procedures).
- The design is exception-driven; the emphasis is on the extensibility and not core commonalities.

The development unit is unable to identify an architect for the system.

4.2.5 Reporting Issues

The final review activity is to create and present a report in which all of the issues, along with supporting data surfaced during the review, are described. The report should be circulated in draft form to all of the review participants to catch and correct any misconceptions and biases and to correlate elements before the report is finalized.

4.3 Evaluation Output

The output of an architecture evaluation should minimally include a report that describes a set of ranked issues (project threatening, major, and minor) determined during the evaluation. Other output that is created by virtue of the evaluation process includes the set of scenarios used to evaluate the architecture, preliminary system predications, an identification of potentially reusable components, and enhanced system documentation.

4.3.1 Ranked Issues

The categorized and ranked issues recorded during the evaluation activities are documented formally, along with the data supporting the issue identification. Remember that the evaluation team must determine whether each issue is project threatening (requiring immediate attention before the project can proceed), major (important issue with the architecture that will ultimately have to be corrected in later versions of the system) or minor (not necessary to correct, but advisable for long-term success).

4.3.2 Report

A formal report is delivered to the party or parties sponsoring the evaluation, as well as to the review participants. The report can also contain architectural descriptions and results of individual analyses if they are particularly illuminating or produced as a result of the evaluation and not located elsewhere in the system documentation. The report is usually presented to the project manager or review-sponsoring individual.

4.3.3 Scenario Set

The set of scenarios that was used to exercise the architecture during the analysis becomes part of the system configuration and can be used to analyze an improved system or a related system, as well as to form the basis of eventual test cases. As already indicated, scenarios developed for common systems will, over time, be used to formulate more generalized questionnaires and checklists.

4.3.4 Preliminary System Predictions

Depending upon the criteria that drive the evaluation, certain predictions about the eventual system will emerge. If performance was a key architecture evaluation criteria, it is likely that, as a result of the evaluation, some rough worst-case/best-case performance predictions are available. If system modification was a key criteria, preliminary predictions of the kinds of modifications the system can support will surface. Among these predictions can likely be the set of potentially reusable design structures that could be shared by multiple similar systems.

4.3.5 Enhanced Documentation

The evaluation will precipitate the development of architecture representations, clarified requirements, and system scenarios as described earlier. Having an established architecture evaluation process will push an organization to standardize its representations of this information.

The aggregated output from multiple evaluations leads to courses, training, and improvements to the system development and architecture evaluation processes.

Outputs are delivered as specified in the contract although we recommend delivering the output to the project that was evaluated. The organization will also wish to keep scenarios and architectures from the evaluation as input to an improvement cycle for the evaluation process.

Costs and benefits of the evaluation should be collected. Estimates of the benefits are best collected from the manager of the development.

5 Recommendation Summary

The main recommendations for architecture evaluations are summarized below.

5.1 Conduct a Formal Review with External Reviewers

The major decision that drives many of the recommendations in this report is to hold an architecture evaluation as a formal review with external reviewers.

We recommend the formal and scheduled review because it provides a focus for an architectural evaluation. A review that is included in the schedule and that is known about far in advance by the developers causes the developers to produce particular artifacts such as an explicit architectural representation. Such artifacts are part of the best development practices, but are not always produced in practice. The recommendation for a review assumes that having an explicit system architecture is a key element to a successful development. The alternative is to view architectural evaluation as a portion of the normal development process. This isn't precluded by the review recommendation, in fact it is encouraged, but having a review provides a milestone and a schedule for the evaluation.

External reviewers are recommended for several reasons. They provide an independent perspective uncolored by pride of authorship. They provide a mechanism for importing architectural expertise. An external evaluation team provides a mechanism for developing an organization's architectural expertise and provides management with an additional verification mechanism.

5.2 Time the Evaluation to Best Advantage

The recommendation is to hold the evaluation after requirements have been established and there is a proposed architecture to review. An architecture evaluation cannot be held prior to the existence of a proposed architecture. As described earlier, AT&T has a type of review called an "architecture discovery review" that is really a checking of the reasonableness of the requirements prior to an architectural review, but this type of review answers different questions. If performance is going to be a consideration during the evaluation (a common occurrence), there must be a provisional binding to hardware so that performance questions can be answered.

As time passes in any development effort, decisions are made about the system under construction. With every decision, modifications to the architecture become more constrained and, hence, more expensive. Performing the architecture evaluation as soon as possible reduces the number of constraints on the outcome of the evaluation.

5.3 Choose an Appropriate Evaluation Technique

In Section 3, we enumerated five different evaluation techniques, distinguishing between questioning and measuring techniques. Scenarios are beginning to emerge as a universally recommended technique for organizations beginning to adopt architecture evaluation practices. The other questioning techniques are derived from scenarios as an organization's evaluation practices mature.

Prototyping tends to be an expensive technique and is usually inappropriate just for the purposes of a review. Prototyping is a popular technique in requirements elicitation, but those prototypes usually do not bear any resemblance to the final system and so their architecture is not subject to this evaluation.

Simulation at a medium or coarse grain is the recommended technique for performance, again because of expense. Because of the existence of performance models and simulation systems, developing and analyzing a simulation is not an expensive undertaking.

Metrics may be useful to help understand the modifiability of the proposed design but metrics make assumptions about the types of modifications that are going to be performed and these assumptions and their relevance to the system under review must be understood. In general, the assumptions that underlie metrics are not enumerated explicitly, and so we did not emphasize them as an evaluation technique.

5.4 Create an Evaluation Contract

Since we recommend that the evaluation be done by an external review team, it is important that both sides understand what will and will not be accomplished by the review. A formal contract is merely a mechanism for ensuring that there is mutual understanding of the review's scope.

5.5 Limit the Number of Qualities to Be Evaluated

All reviews end up covering a great deal of territory and end up discussing management issues, such as cost, schedule, and resource allocation, in addition to purely technical issues. By focusing on a limited number of qualities, agreed to in advance, the evaluation team can be sure that the development organization has its priorities in order and that all parties understand those priorities. When issues arise that are out of scope, since the qualities to be investigated are explicit and have been publicized, it is easy to bring the review back on track. Without some type of restriction as to scope, it is possible for the review to wander and not perform the task for which it was designed.

5.6 Insist on a System Architect

One of the imperatives for a successful development effort is that there is one person who has intellectual understanding of the total system (not necessarily in every detail). Fred Brooks writes:

> *I am more convinced than ever. Conceptual integrity is central to product quality. Having a system architect is the most important single step toward conceptual integrity... After teaching a software engineering laboratory more than 20 times, I came to insist that student teams as small as four people choose a manager, and a separate architect [Brooks 95].*

The lack of an architect is an indication that the development effort has problems, and drives the recommendation that the system architect be available for the review.

Finally, there is a distinction between an architecture evaluation as an assessment of an architecture-based development process and the formal review we recommend. This distinction is made explicit in the questioning process for architecture reviews at Siemens. The evaluation we describe determines whether the given technical solutions are good. Starting from first principles, this evaluation will ask questions to determine what the architecture should be and whether the steps used in determining the architecture were performed correctly. In contrast, an assessment looks to see whether particular issues have been addressed. An assessment seeks only to gain confidence that important issues were reasonably considered by the design team. An assessment can be used as a filter to determine whether or not to do a formal review.

References

[AT&T 93] AT&T. “Best Current Practices: Software Architecture Validation.” Internal report. Copyright 1991, 1993, AT&T.

[Boloix 95] Boloix, Germinal & Robillard, Pierre N. “A Software System Evaluation Framework.” *Computer 28,* 12 (December 1995): 17-26.

[Brooks 95] Brooks, F. P. Jr. *The Mythical Man-Month: Essays in Software Engineering (20th Anniversary Edition)*. Reading, Ma.: Addison-Wesley, 1995.

[Henry 81] Henry, S. & Kafura, D. “Software Structure Metrics Based on Information Flow.” *IEEE Transactions on Software Engineering* SE-7, 5 (September 1981): 510-518.

[Joseph 94] Joseph, Sujoe & Sisti, Frank. *Software Risk Evaluation Method, Version 1.0*. (CMU/SEI-94-TR-19, ADA290697). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1994.

[Kazman 94] Kazman, R.; Bass, L.; Abowd, G.; & Webb, M. “SAAM: A Method for Analyzing the Properties Software Architectures.” *Proceedings of ICSE 16*, Sorrento, Italy. (May 1994): 81-90.

[Kazman 96] Kazman, R.; Abowd, G.; Bass, L. & Clements, P.; “Scenario-Based Analysis of Software Architecture.” *IEEE Software 13*, 6 (November 1996): 47-55.

[Kruchten 95] Kruchten, P. B. “The 4+1 View Model of Architecture.” *IEEE Software 12*, 6 (November 1995): 42-50.

[Selby 95] Selby, Richard W. & Reimer, Ronald M. “Interconnectivity Analysis for Large Software Systems.” *Proceedings of the California Software Symposium.* California. (March 1995): 3-17.

[Smith 94] Smith, C., “Performance Engineering,” 794-810. *Encyclopedia of Software Engineering*, Vol. 2. New York, NY: Wiley, 1994.

Appendix: Workshop Questionnaire

Software Architecture Evaluation

Organization _____
Presenter _____
Any Special Name for Evaluation Approach _____

1. Overall Context and Motivation for Evaluation

1.1. Why is evaluation done?
List technical rationale as well as any internal or external motivators.
1.2. What are the anticipated benefits of the architecture evaluation?
1.3. Are these benefits quantified?
If so, how?
1.4. Who are the customers for the evaluation?
1.5. Other comments on context and motivation.

2. Preconditions (what is necessary for an evaluation to take place)

2.1. What are the evaluation criteria?
2.2. How are the evaluation criteria expressed, chosen, defined, and represented?
2.2.1. Who derives the evaluation criteria?
2.3. What architecture representations (language, model, etc. including semantics) are used for the evaluation purposes?
2.4. What are the personnel requirements?
2.4.1. How many evaluators are needed?
2.4.2. Are the evaluators a special team or part of a development team?
2.4.3. List any training needed to perform and/or use the results of the evaltions.
2.5. Are there any system requirements such as specific purpose, expected life span, operating environment?
2.6. Describe the necessary corporate support such as resource allocation, company policy, organizational infrastructure.
2.7. When in the life cycle is the evaluation performed?
2.7.1. When is the earliest it can be performed?
2.7.2. When is the latest it can be performed?
2.8. Other preconditions.

3. Evaluation Activities

3.1. Describe the methodology, particular techniques, and defined processes that are part of your evaluation approach.
3.2. Describe any tools that are used in the evaluation. Indicate if they are standard tools or were specially constructed in-house.
3.2.1. Are tools necessary for your approach?
3.3. Is the evaluation approach tailorable to either less or more thorough evaluations?
3.4. Other comments on evaluation activities.

4. Evaluation Outcomes (Postconditions)

4.1. What is the output of the evaluation?
4.2. How is the output validated?
4.3. Is the output quantified?
If so, state how.
4.4. How is the output communicated?
4.5. Who uses the evaluation results?
4.6. What is the follow-up on evaluation results?
4.7. How is the output fed back into the development or acquisition process?
4.8. How often is the evaluation performed on a given project?
4.9. Other postconditions.

5. Experience

5.1. Where has the evaluation approach been used?
5.2. Which specific domains?
5.3. Which specific system types?
5.4. For what size systems?
5.5. How many projects have used the evaluation approach?
5.6. Is the same evaluation approach used for every project?
5.7. Has the evaluation approach been improved over time?
Describe any process for continuous improvement of the approach.
5.8. What have you been able to reuse in successive evaluations (for example, scenarios, report forms, people)?
5.9. What has been the added cost of the evaluation approach to each system?
5.10. Have you weighed costs versus benefits?
If so, indicate the results.
5.11. Describe any changes in your software development (or acquisition) process that have resulted from the evaluation activities?
5.12. List any other tangential benefits from the evaluation activities (for example, corporate history, promotions, etc.).
5.13. List the factors that you consider critical to the success of the evaluation activities.
5.14. Other comments on experience with the approach.

6. Other

Please add any additional comments that are important to the understanding and analysis of the evaluation approach.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS
None

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for Public Release
Distribution Unlimited

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
CMU/SEI-96-TR-025

5. MONITORING ORGANIZATION REPORT NUMBER(S)
ESC-TR-96-025

6a. NAME OF PERFORMING ORGANIZATION
Software Engineering Institute

6b. OFFICE SYMBOL (if applicable)
SEI

7a. NAME OF MONITORING ORGANIZATION
SEI Joint Program Office

6c. ADDRESS (city, state, and zip code)
Carnegie Mellon University
Pittsburgh PA 15213

7b. ADDRESS (city, state, and zip code)
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

8a. NAME OFFUNDING/SPONSORING ORGANIZATION
SEI Joint Program Office

8b. OFFICE SYMBOL (if applicable)
ESC/AXS

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F19628-95-C-0003

8c. ADDRESS (city, state, and zip code))
Carnegie Mellon University
Pittsburgh PA 15213

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT NO.
63756E	N/A	N/A	N/A

11. TITLE (Include Security Classification)
Recommended Best Industrial Practice for Software Architecture Evaluation

12. PERSONAL AUTHOR(S)
Gregory Abowd, Len Bass, Paul Clements, Rick Kazman, Linda Northrop, Amy Zaremski

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (year, month, day)	15. PAGE COUNT
Final	FROM TO	January 13, 1997	43

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (continue on reverse of necessary and identify by block number)
FIELD	GROUP	SUB. GR.	software architecture, software architecture analysis, software architecture evaluation, industrial best practice

19. ABSTRACT (continue on reverse if necessary and identify by block number)

Architectural decisions have a great impact on the consequent quality of software systems. As a result, it is important to evaluate how a software architecture meets its quality demands. Though much focus has been placed on modeling and describing the software architecture as a design artifact, we found that relatively little is known about the current experience with software architecture evaluation.

This report details the results of two workshops on software architecture evaluation, held at the Software Engineering Institute (SEI) on November 9-10, 1995 and May 9-10, 1996. The purpose of the

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ■ SAME AS RPT □ DTIC USERS ■

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified, Unlimited Distribution

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (include area code)	22c. OFFICE SYMBOL
Thomas R. Miller, Lt Col, USAF	(412) 268-7631	ESC/AXS (SEI)

ABSTRACT — continued from page one, block 19

workshops was to determine the state of industrial practice in the evaluation of software architectures with respect to a set of desired quality attributes, and to uncover recommendations for best practices. In this report, we summarize the findings of the two workshops, define a set of dimensions to characterize various software architecture evaluation techniques, and make concrete recommendations for implementing architecture evaluation practices.