

**Best Current Practices:  
Software Architecture Validation with Mappings to MBASE  
and CS577**

Copyright 1991, 1993 AT&T

Copyright 2002 USC-CSE

All rights reserved

Printed in U.S.A.

## About This Book

<b>Purpose</b>	To discuss what is an architecture review board meeting and how to conduct one.
<b>Audience</b>	This document is intended to help managers of software projects and software development organizations to appreciate a strategy for ensuring a successful project.
<b>References</b>	This document was originally created by AT&T to discuss software architecture validation in an architecture review board meeting. USC-CSE has expanded the purpose of an architecture review board meeting to include other aspects of the project that are factors in having a successful project.

## Chapter 1. Introducing Software Architecture Validation

The Best Current Practice, Software Architecture Validation, is the use of a systematic and disciplined approach to reviewing and evaluating the components and interactions of a software design before and during development to detect and resolve potential problems.

The architecture of the system will be reviewed and discussed in a meeting called the Architecture Review Board Meeting. The purpose of this review meeting is to make sure that the project is ready to go to the next life-cycle milestone. This determination however will be made based on several additional factors besides architecture.

The other aspects of the system that will be reviewed as well includes, but not limited to: system overview, requirements and features of the system, process/data flows, external interfaces (prototype), performance and capacity characteristics, operations, administration, and maintenance (OA&M), schedule, cost, and feasibility rationale.

This chapter introduces software architecture validation. This document discusses the purpose and timing of software architecture validation and reviews, costs, and benefits associated with software architecture validation, and lessons learned from the first reviews conducted by the Architecture Review Board. In addition, this document discusses how an architecture review board meeting is conducted, what topics are discussed in the meeting, and how a project team should prepare for the meeting.

Costs associated with correcting system problems increase significantly as project development progresses. Early detection of flaws offers the greatest opportunity to improve the design economically and effectively. Software architecture validation and architecture review board meetings are important tools for discovering problems early in the project cycle.

*Architecture* is defined as “the science, art, or profession of designing and constructing any framework or system. It is the blueprint of the software/hardware system. The architecture describes the components that make up the system, the mechanisms for communication between processes, the information content or messages to be transmitted between processes, the input into the system, the output from the system, and the hardware that the system will run on.

**What is architecture?**

Software architecture analogies are often drawn from the construction profession, since these architects have a much better track record in completing complex jobs on time and on schedule. (There are many more unfinished software projects than uncompleted buildings.)

The definition of architecture is fundamental to the definition of quality: It is designing a solution the customer wants (and will need in the future), at a cost the customer is willing to pay, and on a schedule that satisfies the customer’s need.

System simplicity should be the overriding principle: if there are too many unique processes, too many different IPC (interprocess communications) mechanisms, or nongeneric or undefined message sets, the system is likely to have problems. Following *standards* reduces the cost of development and improves the future flexibility of the product. Standards include off-the-shelf components (both system and application components), well-understood IPC mechanisms, and sound architecture approaches. If the topology of the process structure cannot be drawn on at most two pages – or if the interconnection of processes looks like spaghetti – the architecture is probably too complicated.

**What makes a good design?**

Architecture is not unlike the blueprint for a building: if your building does not use standard components and specifications (for example, 12” centers between the studs instead of 16”) or does not have perpendicular walls, you can expect it to be significantly more costly than one in which standards are employed.

Simplicity in the architecture means easier testing and debugging of the code, greater flexibility to support customer-generated changes, and a more robust system for the customer. It also spawns an organizational structure that minimizes overlap and confusion.

The goal is to produce high-valued customer features in the shortest possible interval, while remaining extensible for future enhancements, and providing a system that can be easily maintained.

System design can be viewed as a series of steps:

- **Prospectus:** Definition of the problem and the rationale for doing the project – the economic justification and the overall benefit. Also called the *business case*.
- **Requirements:** Features and functions of the system defining scenarios for usage (forms/screens, workflow, impact on existing environment, etc.), layout of specified reports, algorithms, specifications, etc.
- **Architecture:** High-level configuration of hardware, software, and human factor issues of a system or product to effectively support feature content. It defines the environment to support features such as process structure/data flow, major components, interfaces, and design strategies.
- **High-level Design:** Identifies the critical issues addressed in each of the major components, and specifies the design strategies.
- **Low-level Design:** Identifies the internal decomposition of each module and the functions/procedures in it.

**When is the architecture completed?**

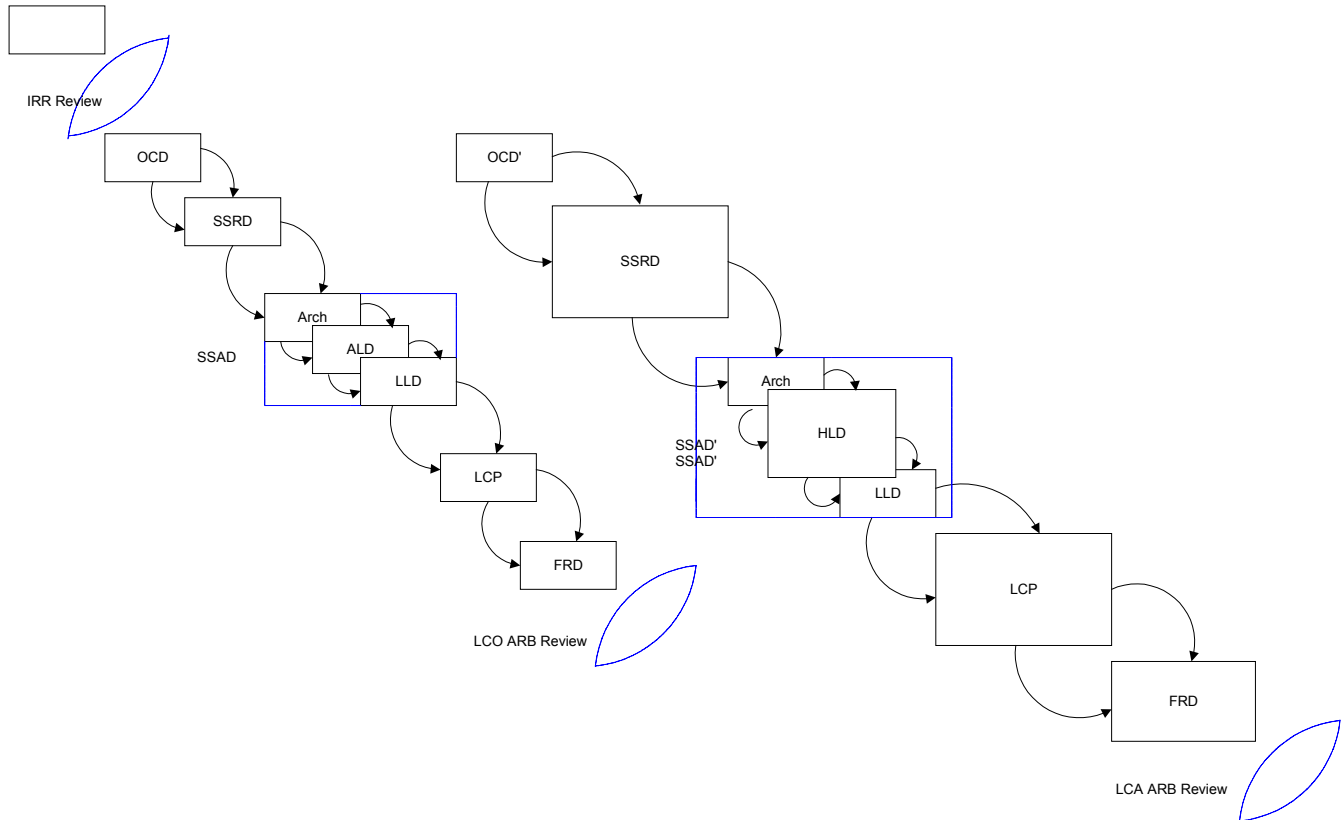
These represent the first steps in the traditional waterfall diagram that is often taught in software project management workshops. They are sometimes viewed as sequential steps. However, to come up with a consistent, cohesive architecture, several steps (Requirements, Architecture, and High-level Design) should be carried out in parallel. This makes it possible to evaluate the impact of the requirements on the architecture, and the feasibility of implementation, before any of the corresponding documents are *baselined*.

Architecture review is an adjunct to formal review processes that projects are currently using, such as project management audit and design review.

## Formal Reviews

### When is the architecture validated?

Most projects have at least two points at which validation should occur, as shown in the following diagram.



*The Inception Readiness Review (IRR)* is optionally held as soon as the following conditions have been met: 1) business case has been prepared, 2) the candidate system objectives, scope, and boundary has been identified 3) the stakeholders have been identified and 4) resources have been committed to achieve a successful LCO package. This review also explores potential architectures that might be appropriate for solving the problem. The *Life Cycle Objectives Review (LCO-ARB)* is held after the project team has baselined the requirements, architecture, and high-level design--that is, when the team has settled on at least one possible approach to solve the problem. In addition, during the review, the project's candidate architectures are shown to be feasible in regards to the requirements, budget and schedule in the life cycle plan. In addition, major risks have been identified in the LCO-ARB. From the review, a final LCO package should be produced

outlining the current state of the project. The *Life Cycle Architecture Review (LCA-ARB)* occurs when the project team has selected a feasible architecture. During the review, the feasibility of the architecture is validated, the stakeholders commit themselves to the Construction, Transition, and Maintenance phases, and all major risks are resolved or covered by a risk management plan. From the review, a final LCA package should be produced which includes Quality Management and Transition plans.

---

**Who  
validates the  
architecture  
[project]?**

To obtain feedback on the project from an objective, unbiased source it is important to schedule reviewers who are from outside the project. Preparing for and conducting effective software architecture validation reviews requires the participation and support of project team members, the Architecture Review Board, the client, and other interested project stakeholders that has been identified in the Operation Concept Description (OCD). The responsibilities of the participants are described in Chapter 2, Roles and Responsibilities.

Inception Readiness, Life Cycle Objectives, and Life Cycle Architecture reviews should not replace other formal review processes in a project, either before or after the high-level design. Architecture reviews should be viewed as an adjunct to other formal reviews.

---

**Inception  
Readiness  
review**

The front end requirements and design process begins with architecture [project] discovery, a series of meetings held with project members and client to increase reuse of processes, ideas, and software to reduce project risks, costs, and intervals. An EasyWinWin negotiation can be held to identify what are the win conditions of each of the project stakeholders. The win-win conditions, in turn, will be used to formulate the project requirements. Even when projects are effectively reviewed at all stages of development, the most valuable feedback comes at the early stages: the closer to the initiation of the project, the greater the payback.

One place to consider other reviews is during system/load test when the system fails to meet the performance goals in the requirements and outside assistance is needed to explore alternatives for better performance.

Inception Readiness reviews also help a project to better estimate schedules.

The timing of the IRR largely determines how effective it will be: the earlier in the project, the more time there is to take action based on the results from the review.

**Architecture reviews**

The first *formal* architecture review (LCO-ARB) is conducted when the Planning and Architecture Team has completed a proposed architecture(s) or solutions. At this stage the high-level project requirements should be complete, an architecture document with possible architectures written, and the high-level design documents completed for the major functional components of the system.

A second formal architecture review (LCA-ARB) is conducted when the Planning and Architecture Team has decided on a final architecture to meet the project requirements. At this stage all the requirements should be complete, an architecture document written with a proposed final and feasible architecture satisfying all project requirements, and the high-level and low-level design documents are completed for all the functional components of the system. In addition, transition plans are made as well.

A subsequent review called the Core Capability Demonstration (CCD) should be held after development is underway and some initial performance measurements are available from the system. This review may last only a half- to a full day, and examines some specific performance issues and to make sure the core capabilities of the system are met. Ideally, some members from the original review team should be present.

After the system has been fully developed, a Transition Readiness Review (TRR) should be conducted. At this stage all software development is complete, site preparation for the system has been done, and support, training, and system operational documentation has been written.

The results of architecture reviews should be reevaluated in each project review meeting. This will help the Planning and Architecture Team to continue to focus on the issues raised by the reviews, and will help it to do a self-evaluation to recognize any new issues that *may* require action as a result of the project review meeting.

---

## Costs and Benefits

---

### Costs of validating architecture

On an average industry project, it is estimated that a review will require a total of 70 staff-days of project effort (distributed across about 15 people):

- 30 staff-days of preparation time by the project team
- 30 staff-days during the two days of presentations and readout (assuming entire project attends the review)
- 10 staff-days to prepare for the ARB meeting (address areas of concern, create action plans, etc.)

*The effort is substantially reduced if the project can reuse most of its existing documentation in preparing for and presenting the review.*

### **CS577 Guidelines:**

On an average CS577 class project, it is estimated that a review will require about 2 weeks of time from each of the project team members. For the ARBs, the following things must be completed:

- Two weeks before ARB:
    - Arrange time slot with client
    - Discuss ARB process and issues with clients
  - One week before ARB:
    - LCO/LCA drafts on Web
    - Dry run of presentations and demo
  - ARB Week:
    - ARB Presentation and discussion
    - Follow-up team discussions, client discussions
  - One week after ARB
    - Final LCO/LCA packages due
-

**Benefits /  
Lessons  
Learned**

- Project teams found that the preparation for the review helped them get a clearer understanding of their projects. Often, this was one of the few opportunities for the project team members to have in-depth discussions of the technical issues about the project. The preparation served as a catalyst to bring them together.
- The ARB ensured good architecture and commonality and to make sure project requirements were satisfied.
- The ARB checklist helped formalize and standardize the design process. The checklist should be a list of questions that the projects stakeholders help refine. Chapter 3 provides an initial set of questions the stakeholders may look at to help them get started.
- The reviews have become an important forum for exchanging valuable information and for identifying reuse opportunities. It is as much a learning process for the reviewers as for the reviewees. The reviewers take back to their projects insight into alternative approaches to a solution.
- Earlier involvement of the ARB in the design has led to better recommendations and more positive influence on the architecture.
- The reviews promote operations, administration, and maintenance (OA&M) consistency and completeness.
- Project teams are motivated to pay closer attention to the performance and error recovery implications of their architecture.

## Lessons Learned

---

**Overview** Architecture reviews conducted by the ARB have detected design problems in the areas of requirements: performance; operations, administration, and maintenance (OA&M); and error recovery. This section discusses these design issues.

---

**Requirements** Some projects fail to appreciate the fundamental high-valued functions required in their products. During the design phase of the system, the architecture should be evaluated against these high-valued functions, but in some cases the architecture is designed without a clear understanding of the problem the product is aimed at solving. Getting a better fit between the product goals and the architecture is best accomplished by a small Planning and Architecture Team made up of key systems engineers, business planners, architects and developers who can best represent the customer needs and the implementation possibilities. Projects that have used such a team have found that iterating both the requirements and architecture with the customer has helped improve the design. In addition, use of CASE tools effectively allows a concise way to represent both requirements and design to manage change control.

---

**Performance** Performance headroom is one of the major areas of concern in reviews. Projects often select hardware before the system architecture has been completed, resulting in later performance problems. Techniques to analyze performance include scenario testing, tracking transactions from source to sink, and estimating or counting the CPU, I/O, and database updates/queries performed by each operating system process. The following discussion describes performance-considerations for the UNIX\* system processes. Building a simple spreadsheet of these scenarios allows the project to ask 'What if' questions, such as, what happens if the number of input records grows by 20%, or if the transaction rate doubles.

---

\* UNIX is a registered trademark of UNIX System Laboratories, Inc.

Performance considerations requiring special attention include:

- Writing the same data multiple times to the disk
- Gracefully handling extreme bursts of data when restarting the system
- Handling peak loads as well as average loads
- Being flow-controlled by external systems
- Allowing for expected load growth as functions are added, deployment is increased, or more centralization occurs than expected
- The system resources needed for functions such as process switching and file system lookups.

Projects should understand the penalty of the system losing data and the tradeoff of servicing high priority functions at the cost of lower priority functions or loss of less critical data.

Prototypes of the critical thread of an application can be used to identify performance "hot spots." In one project, the prototype showed a critical process executed an order of magnitude slower than expected, and the project was able to redesign the functions. Prototypes can also be used to measure critical interprocess communications or the UNIX® system resources required to implement the function.

Projects should allocate a resource budget to the functional components (processes in a UNIX® system application) of their architecture and then *track* the usage of these resources as the developers are doing their unit testing. Part of the system test plan should be to verify that each component is operating within its budget.

Systems should be designed with built-in instrumentation that can be turned on to measure field performance under 'real' loads. The elapsed time to make the backup media should be calculated by counting the number of records times the media write time to make sure that the backup can be completed in the allowed interval.

---

## OA&M

In many projects the design of the system operations, administration, and maintenance is the last concern. This has resulted in awkward, error-prone system operation. Designers in these projects need to better understand the customer's expected operation of the system. Many requirements fall short in this area.

Several projects planned rapid deployment to hundreds of sites over a two-year schedule, but did not have automated delivery mechanisms. Other projects deployed with non-UNIX® system personnel but still used

the UNIX® system startup mechanisms. Deploying systems internationally may require special operations and/or installation requirements.

Backup/recovery of critical data is often considered an important feature, but in practice the field does not rehearse the recovery procedure often enough to use it effectively when needed.

Most projects do error-logging to a single standard event/error logger (for example, BaseWorX\* Event Log) with browsers and standard reports for output.

There is a growing need for sophisticated security to prevent unauthorized access to the system and partitioning both data and program access by job type. Security mechanisms can be abstracted out of the particular application, providing an excellent reuse opportunity for projects.

*Control mechanisms* are also needed to allow tuning the system dynamically, providing overload prevention, or statically partitioning resources among several processes. Some projects have implemented *monitors*, daemon processes that watch high water marks on message queues, file systems, or databases and take appropriate recovery action. A process monitor can be used to restart failed processes. With UNIX® for example, the use of TUXEDO/T software provides both process monitoring and message queue management.

Some projects have not accommodated a system design that would allow multiple instantiations of the system. These projects need to force both the installation process and the startup process from starting a second copy of the system. In any case, systems should be designed without built-in device names, file system names, or even process names (numbers). The system should be designed to abstract these parameters using the operating system environment variables or files containing the names so that alternative copies of the system can be booted, if only for debugging. UNIX® system environment variables are easy to implement and very convenient to change.

Product designers should understand:

- Deployment plans
- Installation strategy

---

\* BaseWorX is a trademark of AT&T.

- Security needs
  - Backup/recovery criteria
  - Impact of loss of data, processes
  - Impact of system outage and length of outage
  - Resource management needs
  - Expected operating environment, number and level of operators
  - Robustness of requirements.
- 

**Error recovery**

Error recovery is an aspect of OA&M that needs special attention because of the increasing complexity and distributed nature of our products. Several projects have converted from single-processor operation to multi-processor configurations. Other projects have moved more processing into distributed workstations. Many products are networked together and are expected to operate as a single system.

Error recovery in these configurations requires remote detection of process failure, data loss, etc., and then recovery actions across two or more disjoint processors. For efficient operation, error logging or error reporting needs to be consolidated for the operator. Synchronization of data or databases, displays, and software updates becomes critical for robust performance.

Successful projects implementing distributed systems have used symmetry of design and standard networking to improve the error recovery. Wherever possible, the software in the remote processors is identical (either could be the sender or receiver), the processes are restarted locally automatically, the connections are reestablished automatically, and all processes are prepared to be flow-controlled (usually implemented by a system-wide library that handles all IPC). The automatic mechanisms may require implementing extra connection monitors or daemons, but eliminating human interventions will improve non-slop operation. Automatic mechanisms also mean that graceful shutdown can occur during failures without damaging data or databases or leaving around dangling items such as lock files.

Most distributed projects implement local error logging with a mechanism to merge the logs at reporting time.

**Transition**

The project team and the client should be thinking about the Transition and Maintenance phases early in the life-cycle plan. A maintainer for the system should be identified as well as a target server for the delivered system as soon as possible.

---

---

Before transition begins however, it is suggested that the team develop the system on a server/platform similar to the target server/platform the system will be run from in order to reduce some of the risks faced during Transition.

---

**Risks**

A purpose of the ARBs is to help the project team and client identify risks in the project. If a risk is not identified early in the life cycle, it can push the schedule back or cause the project to fail. In addition to the ARB, the project team should try to identify risks in the project on a weekly basis so as to ensure the project continues smoothly. A good reference to help identifying risks is the Top 10 Risks list developed by Dr. Boehm.

---

**Schedule**

When negotiating with the client on the system requirements, the team and client should take schedule into account. Since the project team only has 12 weeks in the Construction phase, with two of those weeks dedicated to the Transition phase, the project team and client should come up with a set of requirements that is feasible to develop based on the schedule of the course and the resources of the team.

In addition, the project team should work with the client on identifying what are the core requirements of the system. Therefore, if problems were to arise during the Construction stage, the project team could at least deliver the core system to the client at the end of the Construction phase.

---

**Maintenance**

To ensure that ISD will maintain the system after transition, the project team should work with ISD on quality management for the system. For example, coding standards should be set up between the maintainer and the project team as well as any other plans/standards/methods that will help the maintainer with maintaining the system. In addition, training should be held with the maintainer after transition to help him/her become familiar with the system.

As early as possible in the life cycle, the project team should work with the client to identify a possible maintainer for the system based on what skills the maintainer will need. Identifying a maintainer early in the life cycle will help the team construct a better quality plan and will ensure a smoother transition and acceptance of the system on the client's end.

---

**Conducting Formal Software****Architecture Validation Reviews**

---

Software architecture validation is a team effort to ensure that the system architecture is a reasonable and cost-effective solution to the problem. This chapter

- Identifies participants and their responsibilities
- Describes the documents that are the basis of the review: the LCO and LCA package along with the prototype
- Details the review process
- Provides recommendations for presenting and using results from the review
- Describes pitfalls and provides suggestions for avoiding them.

## Roles and Responsibilities

---

**Overview** Preparing for and conducting effective reviews for software architecture validation requires the participation and support of the organization manager, the project manager, project team members, the Planning and Architecture Team, and the Architecture Review Board. This section describes the responsibilities of these participants.

---

**Management** Organization Manager: The organization manager provides leadership and resources for creating and validating system architectures for all projects in the organization

Project Manager: The project manager ensures that the customer's needs are met with the *simplest* design possible. The manager should identify the high-valued customer features and walk-through implementation scenarios using the architecture. The architecture review will assist the project manager in evaluating the design.

CS577 Guidelines:

In CS577, management will be the CS 577 teaching staff. The staff will help you validate your architecture in the ARB meetings.

---

**Project team** The project team creates the LCO/LCA packages and prepares for presentations to the ARB, using the checklists in Chapter 3 as an aid to assemble all relevant system design information. On some projects, the project team members maintain a diary on-line summarizing their daily activities, so other project members can scan the file for an update on the project status.

---

**Planning and Architecture Team** Project teams do not always understand the fundamental high-valued customer features required in their products. Getting a better fit between the product goals and the architecture can best be accomplished by a small Planning and Architecture Team. The Planning and Architecture Team is a subset of the project team, with specific responsibility for developing the system architecture. This team's primary responsibility will be to write the SSAD and LCP documents.

**Members**

The team may include representatives from the following areas of expertise who can best represent customer needs and implementation possibilities.

- Product management/marketing
- System engineers or analysis (responsible for requirements generation)

*Note:* “System engineer” is not a universal term in the industry. We use it to specify those who write system requirements; the term “analyst” is also used.

- Software/hardware developers
- System test/support
- User/customers.

**CS 577 Guidelines:**

In CS577, your client will attend the ARB meeting to ensure the system meets their needs. In addition, it is recommended that the maintainer of the system attend the meeting as well so that the maintainer can see how the system is going to be developed and whether or not the system is understandable enough for him/her to maintain. In addition, a test user of the system would be helpful to have at the meeting so that he/she can comment on the prototype and how the system will be used.

- Chairperson** The chairperson for this team can either be selected by the team or designated by the management. This person is responsible for:
- Getting the team together to review the current status and documentation
  - Acting as a *facilitator* in helping the team reach consensus. The person is typically either a developer or a system engineer.

CS 577 Guidelines:

The chairperson for the 577 ARBs will be Professor Boehm. Professor Boehm and the 577 staff will make up the board that will evaluate your project and act as a facilitator between the client and the project team.

**Approach**

The process of defining the architecture is an iterative one. The Planning and Architecture Team continually reviews both the high-level requirements and system design. The most important job of the architects is to ask questions about the system. Architects need not know the answers to all the questions, but must be catalysts for others to find the answers. The Planning and Architecture Team must constantly ask itself:

1. What is the problem we are trying to solve?
2. How will people use this system?
3. How can we simplify the architecture?
4. Is it buildable within the schedule and budget?
5. Have all associated risks been answered or identified?

This approach allows the system engineers and designers to review the architecture and requirements with the customer to ensure the system design is the best approach to meeting the customer's needs. A clear understanding of the customer's perception of the problem is essential to designing an architecture and delivering a product that will solve the problem. Regular review of requirements and architecture along with the project schedule with the customer enables the team to improve the design and project schedule. Furthermore, an agreement with the client on what are the core capabilities of the system should be discussed so that if time were to run out, at least the core system can still be delivered.

Each representative has input to the process that defines (and is responsible for) the architecture. The Planning and Architecture Team determines requirements on the system, constraints on the environment in which it is to execute, specific hardware that it is to support, user interfaces that are to be provided, etc.

The resultant architecture is the team's consensus view of a reasonable solution.

***Lifespan***

The Planning and Architecture Team is formed at the inception of the project, after a business case/prospectus has been put together. The knowledge contained in this team must be available throughout the life cycle of the project; that is, the makeup of the team may change over time, but the information relating to the reasons that decisions were made must be passed along from one generation to the next. The Planning and Architecture team leader maintains a diary of decisions made on the project and distributes it regularly to the other project members.

This team spends several days (off premises, if possible) defining the constraints on a potential solution, initial set of high-level requirements, assumptions on possible approaches, and laying out a preliminary architecture. At this point a IRR is conducted. Based on the output of the IRR, the team continues working together to produce the requirements, several feasible architectures, and high-level design document. This process may take several weeks to months, depending on the size of the project. At this point an architecture review is done on the project.

At the LCO review, the project will examine the architectures and ensure the team has a firm grasp of the project requirements. Based on the output of the LCO review, the project then begins the LCA part of the life cycle plan. Once the Planning and Architecture Team selects a final architecture it believes will satisfy all project requirements, a LCA-ARB meeting will occur to ensure the architecture is feasible and all stakeholders' win-win conditions are satisfied. In addition, the LCA-ARB will look at plans the project team has set up for the Construction and Transition phases.

Based on the output of the LCA review, the project begins the implementation phase. The Planning and Architecture Team must continue monitoring the project to ensure conformance to the requirements and architecture.

---

**Architecture  
Review Board**

The Architecture Review Board typically consists of three to five subject matter experts. A chairperson is appointed to lead the team, and meets with the project manager to determine the areas of expertise required for the review. The project manager may request specific reviewers who have knowledge about the system.

The chairperson selects the review team (the ARB maintains a database of experts). The review team reads the design documentation (requirements, architecture, and high-level design), generates a list of questions for the project team, and uses both documents and responses to develop a project evaluation that specifically identifies strengths and weaknesses of the project's architecture.

---

CS 577 Guidelines:

In the course, the 577 staff will make up the Architecture Review Board.

---

## Project Design Documents

---

### Overview

The basis of the validation Process is the Project design documentation. This section describes the documents that are Provided to the ARB as a first step in preparing their evaluation.

---

**LCO/LCA package**

The LCO/LCA packages made up of 5 documents that includes the OCD, SSRD, SSAD, LCP, and FRD addresses the following items:

- One or two paragraphs (i.e. the Elevator Talk) stating the problem that is being solved (the *what, why, and scope of* the problem)
- Enumeration of the major features (hardware and software) that the system will provide (bullet list of high-level requirements) and/or prototype
- Assumptions and constraints that will bound the solution
- High-level data flow/process diagram of a proposed solution/architecture.
- *Performance parameters* relating to critical events that occur in the system
- Current major risks identified with the solution.
- Life Cycle plan
- Feasibility Rationale: Business case

The OCD document describes overall context of the system to be developed, why it's being built, what exists now, and where the project is starting from.

The SSRD document describes the requirements of the system and project.

The SSAD or architecture document describes the selected approach, and is used by members of the project to ensure that their activities are consistent with the major direction of the project. It is made available to members of the review team before the architecture reviews.

The SSAD is an *adjunct* to the existing documentation (for example, requirements, prospectus, etc.) on the project. For some projects, it will be a hierarchy of documents, comprised of a top-level description of the system with separate architecture documents for the subsystems.

The LCP document provides a life cycle plan for the project and serves as the basis for controlling the project's progress in achieving the software product objectives.

The FRD document ensures feasibility and consistency of other LCO, LCA package components and demonstrates the viable business case for the system.

---

---

**Diary document**

The *diary document* contains a record of decisions made about the architecture. It documents the reasons why a particular alternative was chosen and, more importantly, why an alternative was not chosen. It provides a history (audit trail) of the project and is invaluable to people who join the project later and want to understand the current status of the project.

The *diary* is not typically maintained by most projects but should be considered a valuable part of the audit trail to determine why design decisions were made. This aids people coming into the project at a later date to understand what the conditions were at the time the original decisions were made.

The Planning and Architecture team leader is responsible for maintaining the diary.

CS 577 Guidelines:

Generally not done in CS 577.

---

**Debriefing document**

When the project (or generic) is complete, the diary is published as a *debriefing document* so other project teams can learn from the experiences, both positive and negative. The debriefing document contains the following information:

- Things that were done especially well that may apply to other projects (for example, design, programming, or other techniques that were innovated, refined, or improved upon; or simply existing techniques that were used with particular success)
- Things that should have been done differently
- Problems encountered, and recommendations about how future projects might avoid or solve them
- What was learned regarding planning, architecture, etc.

CS 577 Guidelines:

During the ARB, board members will provide feedback on the good and bad aspects of the project.

---

**Formal Review Process**

---

**Purpose**

The purpose of an LCO review is to:

- Show for at least one architecture, a system built to architecture will:
  - Support the Ops Concept
  - Satisfy the Requirements
  - Be faithful to the Prototype which will be shown as well
  - Be buildable within the budgets and schedules in the life cycle plan
- Show a viable business case
- Ensure that the key stakeholders are committed to support the Elaboration Phase (to LCA)

The purpose of an LCA review is to:

- Show for the selected architecture, a system built to the architecture will:
    - Support the Ops Concept
    - Satisfy the Requirements
    - Be faithful to the Prototype
    - Be buildable within the budgets and schedules in the life cycle plan
  - Ensure all major risks are resolved or covered by risk management plan
  - Ensure that the key stakeholders are committed to support full life cycle including the Transition and Maintenance phases.
  - Ensure the selected architecture is feasible
-

**Process  
overview**

The actual review is a two-day (80-minute for CS577) meeting of project team members and the ARB Team. The following are the typical steps in preparing for, conducting and reading out the results of the review.

1. The project team Contacts the ARB to schedule an architecture validation review.
  2. The ARB appoints a review team of subject matter experts. (For CS577: review team composed of CS 577 staff)
  3. The project team provides the LCO/LCA packages to the ARB team two weeks before scheduled review. (For CS 577: one day before review meeting)
  4. The ARB team generates a list of questions and concerns several days before the review.
  5. The participants meet for two days of technical presentations and to discuss the pre-review questions. (For CS 577: Does not occur)
  6. The ARB review team documents preliminary findings for the project team immediately after the review.
  7. The project team develops an action plan based on these findings.
  8. The project team and the ARB review team present the evaluation and action plan to the project's management and the Architecture Review Board directors.
  9. For CS577 only: after review, LCO/LCA package gets updated.
-

**ARB  
preparation**

The ARB provides the project team with the following information:

1. Sample review agenda
2. Areas that the project team should cover during the review
3. Checklist of sample questions that may be asked during the review.
4. Sample performance worksheet showing how the project team could organize its Performance-related data
5. Outline of the presentation to the ARB directors and the project/product managers.

**Project team preparation**

In preparing for a high-level review of the architecture, the project team should consider:

1. Defining the software architecture from a process level, with defined message flows between the processes and defined external interfaces for each process. It should describe the topology of the processes, data flows, and control flows.
2. Evaluating the architecture, using checklists in Chapter 3 and considering the following critical attributes:
  - a. Appearance: simple or complex?
  - b. Well-defined state transitions: precise input, action, precise output?
  - c. Error processing: How to recover from lost transactions/process? What error logging is provided?
  - d. Performance: potential bottleneck processes, available headroom, controls/monitors built into the system?
  - e. System administration: start-up, shutdown, maintenance control?
  - f. Requirements: Understanding the limitations and assumptions? (For example, 100 messages/sec in 1989 and 200 messages/sec in 1991)
  - g. Project walk-through scenarios: complete flow analysis from external stimulus through the system and out to external result?
  - h. Technology sensitivity and hardware obsolescence?
3. Quality attributes associated with the architecture: robustness, naturalness (picture fits the problem to be solved), elegance, extensibility, flexibility, simplicity, evolvability, schedule, risks, maintenance, scalability, security
4. The current architecture and the projection from systems engineering for the top two to three major features in the next five years. Does the architecture fit these changes?
5. Prepare a feasible life-cycle plan
6. Ensure all major risks are covered or identified
7. Practice the presentation within the project team, i.e. dry run
8. Ensure LCO/LCA package is consistent

The project members make presentations in the topic area they are responsible for.

**Review  
agenda**

The Architecture Review Board provides a typical agenda to help organize the two days of presentations. The actual agenda should include the following areas, as appropriate.

- System overview
- Requirements and features of the system
- Hardware architecture
- Software architecture overview
- Process/data flows
- Database needs/strategies
- External interfaces, Prototype
- Performance and capacity characteristics
- Operations, administration, and maintenance (OA&M)
- Error recovery strategies.
- Schedule (Life-Cycle Plan)
- Cost
- Feasibility

**CS 577 Guidelines:**

This is a typical review agenda for an ARB meeting in CS 577:

(x,y): (presentation time, total time)

(10,15) OCD. System purpose; current system and deficiencies; proposed new system; system boundary; desired capabilities and goals; top-level scenarios

(10,15) Prototype. Most significant capabilities

(5,10) Requirements. Most significant requirements

(5,10) Architecture. Top-level physical and logical architecture; status of COTS/reuse choices

(5,10) Life Cycle plan. Life cycle strategy; key stakeholder responsibilities

(5,10) Feasibility Rationale. Business case; major risks; general discussion

(0,5) Things done right; issues to address (Instructor)

---

## Presenting and Using Results

---

### **Preliminary results**

At the completion of the review, the reviewers caucus and prepare an initial readout. Covering the following points:

- Summary of positive package items
- Summary of priority ordered areas of improvement
- Opportunities for reuse of components from this project, and other components the project can reuse.

This readout may take the form of vugraphs, or the reviewers may opt to provide a set of engineer's notes detailing the comments that they have made and suggesting some approaches to address areas of improvement.

CS 577 Guidelines:

The ARB members will provide a list of good and bad things about the project to the team and will determine during the meeting whether or not the team should proceed to the next phase in the Life-Cycle Plan.

---

### **Action Plan**

The project team uses information from the readout to create an action plan to address the issues that have been raised. Again, vugraphs may be created, structured to follow the points in the initial set prepared by the ARB.

**Formal presentation**

The formal presentation of the readout to the ARB directors and the project/product management is scheduled for 80 minutes.

This is a typical review agenda for an ARB meeting in CS 577:

(x,y): (presentation time, total time)

(10,15) OCD. System purpose; current system and deficiencies; proposed new system; system boundary; desired capabilities and goals; top-level scenarios

(10,15) Prototype. Most significant capabilities

(5,10) Requirements. Most significant requirements

(5,10) Architecture. Top-level physical and logical architecture; status of COTS/reuse choices

(5,10) Life Cycle plan. Life cycle strategy; key stakeholder responsibilities

(5,10) Feasibility Rationale. Business case; major risks; general discussion

(0,5) Things done right; issues to address (Instructor)

---

**Presentation to project team**

The full benefit of the review is realized only when review results are communicated to the entire project team. This may be a formal or informal team depending on project scope and size.

---

## Avoiding Common Pitfalls

---

**Introduction** Sometimes an architecture review effort may fail to produce the expected benefits to the project. This section discusses factors that significantly reduce the effectiveness of the software architecture validation process.

---

**Factors** The quality of the review results depends upon having the right people doing the right things.

- Management may not be sensitive to the lower level team members' appear critical.

It is important to encourage people to speak out.

- During technical presentations, project team members reporting for the project are people other than those who are actually doing the work.

Project members who present during the reviews must be thoroughly familiar with the project and able to answer questions knowledgeably.

- Results do not accurately capture issues or problems in the project.
- Reviewers feed back what they see on a project, but their view of the project is limited.

Project members should provide accurate information and avoid influencing review's perceptions.

To avoid a perception that the review has limited value, reviewers must probe beyond the information that the project team provides, and ask key questions to obtain accurate information.

- Project team does not respond to key issues identified, so results do not lead to change and improvement.

Review results need to be communicated through the entire project.

---



**Chapter 3 Ongoing Evaluation:  
Design Checklists**

---

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. In the frenzy of development, people often focus on the current critical item and may lose sight of items and issues concerning other parts of the system.

These checklists provide a starting point to ask questions about the system. They do not necessarily cover all aspects of your system, but rather are meant to stimulate thinking about the issues important to your project: many questions may not apply to a project, and some issues that are important may not be covered in any of the questions.

The checklists are not only used to prepare for the project review, but should be used by all project members to review their current work and to identify any new issues that they should be considering.

The purpose of these lists is to stimulate the thinking process that a project should be going through in considering the areas that will impact its architecture.

The Architecture Review Board Checklist addresses three primary areas:

- Error recovery
  - Operations, administration, and maintenance (OA&M)
  - Performance
-

**Coverage Matrix**

Each question in the ARB Checklist has been classified as one of the three major areas of concern (error recovery, OA&M, and performance) that have been seen on the projects reviewed and by the applicable system component (database, I/O, testing, etc.). The coverage matrix below indicates the number of questions in each of the categories.

	Error Recovery	OA&M	Performance
Architecture	2	13	8
Database	8	9	14
Distributed System	6	3	4
Fault Isolation	8	4	2
Fault Tolerant	3	3	1
Hardware	2	1	5
Input/Output	4	5	11
Integrated Processor	1	3	3
Methodology	1	3	2
Migration	6	2	7
Miscellaneous	1	0	0
Modeling	2	3	18
Object Oriented Programming	0	0	1
Perfection Monitoring	4	6	28
Platform	1	10	3
Prototype	2	0	1
Requirements	1	5	1
Reuse	0	2	1
System Administrative	3	10	2
Testing	2	10	2
Training	0	3	0
User Interface	1	4	2
User Programming	0	1	1
<b>**Total**</b>	<b>58</b>	<b>100</b>	<b>117</b>

**Architecture Review Board (ARB) Checklist**

---

**Error  
recovery**

1. Can the System operate without the database? Can input be accepted for later processing when the database is restored? What are the Manual overrides in the system? **[Database]**
2. How complex is the database conversion? **[Database]**
3. How critical is the database? What happens to the System if the database is corrupted? **[Database]**
4. If other systems are dependent upon your system for updating their databases, how do you ensure that the interface between the systems will remain constant as my system changes? Is positive acknowledgment that they received the update required? Should you be able to retransmit prior updates that the downstream systems missed? **[Database, Architecture]**
5. What is the error rate expected in parsing data input by other systems? What is the cost of incorrect data? Is manual handling required to correct the errors? **[Database, Dist. Sys., User I/F]**
6. Do you also have a database unconversion if you have to go backwards? **[Database, migration]**
7. Is the data being sent between systems in a consistent unit of information? (More than 50% of the trouble reports in some systems are related to communications interfaces within them.) **[Dist. Sys., I/O]**
8. Is there error recovery code in a process to clean up messes that it leaves when it detects an error? Are you sure that this code will be invoked? If not, are there external processes that can perform the cleanup? **[Fault Isolation, Methodology]**

9. If the customer has requested fault tolerance, what are the customer's expectations? Will your System provide both hardware and software fault tolerance? What techniques are you using to ensure software fault tolerance? **[Fault Tolerant]**
  10. Can the system automatically detect and recover from single failures? If so, how long does recovery take? **[Fault Tolerant, Fault Isolation]**
  11. Can hardware diagnostics be run on-line? If so, is there something the application can do to accommodate this? **[Hardware, Fault Isolation]**
  12. How does the hardware respond to errors? Does the application need special functions to respond to these errors? **[Hardware, Fault Isolation]**
  13. What is the strategy used in Boehm's approach to analyzing the "risks" in a system? Is it the same as considering the error modes? Where do they differ? What portions can be used? (Spiral Model) **[I/O]**
  14. How quickly can the file system/database be backed up? Can backups be accomplished while the application is on-line? **[I/O, Database]**
  15. How does the IPC mechanism respond to error conditions (buffer/queues full, etc.)? Is there adequate warning so that no information is lost? If a message is lost, how does the system respond? Can manual intervention clear the error? **[IPCs, Fault Isolation]**
  16. Do you need a "three-day weekend" for the conversion? (That is, does conversion take a few hours or more than 15 hours, in addition to the two days allocated to install new software?) **[Migration]**
  17. How long will it take to get the "old" system back on the air? **[Migration]**
  18. Is it a revolution or evolution? **[Migration]**
  19. What data will have been lost in this transition back to the old system? Have the messages been saved so they can be replayed into the old system? **[Migration]**
  20. What is the backout strategy?
  21. What new technology is being used? What are the risks of using it? What is your fallback strategy? **[Migration]**
  22. How is the system (network and processors) administered? Are performance monitoring tools and mechanisms available through the hardware and controlling systems software to facilitate trouble isolation and resolution as well as performance tuning (load balancing)? **[Perf. Mon., Sys. Admin, Fault Isolation]**
  23. What are the failure modes of the software and how are they handled?
-

**[Platform, Fault Isolation]**

24. If the prototype is used for evaluating the user reactions to the interface, does the user understand that this is short term? (Has a contract been written with the user?) **[Prototype]**
  25. Is the prototype being built with the “intent to throw it away,” or will it evolve into the final product? What level of error handling has to be built in? **[Prototype]**
  26. What are the availability objectives for the system and how does the hardware architecture address them? **[Reqmnts., Modeling]**
-

**OA&M**

1. Is there anything that prevents the application from running on a multiprocessor (for example, use of shared memory)? **[Architecture]**
2. How many different environments are there for maintaining the configuration information in the system? How many different command languages does the user have to learn? Is there one central file form which all (most) of the information is generated? **[Architecture, Database, Sys., Admin]**
3. Is the system constantly being evaluated with respect to what portions should be “renewed” as changes are made? **[Architecture, Methodology]**
4. How do you perform a system test of data that is input from a graphics terminal? Does the testing scenario require that a person be part of the testing sequence? Can the input be simulated (in order to facilitate regression testing)? **[Architecture, Methodology, Testing, User I/F];**
5. Can the system be on-line as changes are made to the application software, or is it taken down to install a new release? **[Migration, Reqmnts., Sys. Admin]**
6. Is UNIX® visible to the normal user of the system? If so, what is done to prevent/limit their unrestricted use of UNIX® commands? **[Architecture, User I/F, User Prog.]**
7. Can disk space be recovered while the system is active? **[Database]**
8. If data from external systems is required to keep the database up-to-date, what arrangements have been made to ensure that this data is delivered on a timely basis? What is the penalty if the data is not received? **[Database]**
9. If other systems are dependent upon your system for updating their databases, how do you ensure that the interface between your systems will remain consistent as your system changes? Is positive acknowledgment that they received the update required? Should you be able to retransmit prior updates that the downstream systems missed? **[Database, Architecture.]**
10. Can the system automatically detect and recover from single failures? If so, how long does recovery take? **[Fault Tolerant, Fault Isolation]**
11. If your system needs a specialized driver, will it violate the system integrity that the vendor has provided in its kernel? (For example, Tandem has special code in its kernel to reduce system panics.) **[Fault Tolerant, Reqmnts., Sys. Admin]**
12. Can hardware diagnostics be run on-line? If so, is there something the application can do to accommodate this? **[Hardware, Fault Isolation]**
13. How quickly can the file system/database be backed up? Can backups be accomplished while the application is on-line? **[I/O, Database]**

14. Are the interfaces between processes consistent? (That is, is just a single method being used for the majority of the communications?) **[IPCs]**
15. Can you cut over a feature at a time? **[Migration]**
16. The resource budgets that are allocated to functions must take into account overload situations where the budgets will have to be adjusted to different values; that is, budget will vary with the state of the system. How do functions detect this and make the adjustments? **[Perf. Mon.]**
17. What are reasonable values for the amount of resources that functions/features in the system can consume? Have the responsible developers been informed of these limits? What mechanisms are in place to track them? **[Perf. Mon.]**
18. How is the system (network and processors) administered? Are performance-monitoring tools and mechanisms available through the hardware and controlling systems software to facilitate trouble isolation and resolution as well as performance tuning (load balancing)? **[Perf. Mon., Sys. Admin, Fault Isolation]**
19. Can more than one incarnation of the application run on a single processor? Can other applications run on the same machine? **[Platform]**
20. Does the platform provide the OA&M requirements for the system? **[Platform]**
21. What is the OA&M strategy used in the system? **[Platform]**
22. What are initialization and termination processing? Describe them. **[Platform]**
23. Is there critical sequencing of functions required at startup/shutdown, and how are the functions controlled? **[Platform, Architecture]**
24. What are the failure modes of the software, and how are they handled? **[Platform, Fault Isolation]**
25. Has the software architecture been matched against the current platform standards such as the BaseWorX™? **[Platform, Reuse]**
26. If deficient, what areas need to be enhanced so that these platforms are usable? Should these enhancements be done as part of your project? **[Platform, Reuse]**
27. What is the high-level process architecture (data flows and interfaces)? Describe it. **[Reqmnts., Architecture]**
28. What are the availability objectives for the system and how does the hardware architecture address them? **[Reqmnts., Modeling]**
29. Does your project need a specialized operating system driver? Can the same be done with existing operating features; in UNIX®, for example

- 
- can the same function be done using STREAMS\*? **[Reqmnts., Sys. Admin]**
30. Are tools built into the system support package that will allow dumps to be taken of critical tables while the system is running? **[Testing]**
  31. Is there a day/month/year's worth of data play back through the system? **[Testing]**
  32. Is there a "stop the world" command that will suspend the entire system at a given point so that a snapshot or a dump can be taken, so that the state can be examined? (This is especially important in a real-time multiprocessor system where complex interactions can occur.) **[Testing]**
  33. Does the system test scenario provide for a complete, end-to-end system test? **[Testing]**
  34. How is the regression database handled? (The dates have always been a problem on the input data; can today's date be input via an environment variable?) **[Testing]**
  35. If the system is partitioned, can the output of the system test of one partition be used as input to the next one? **[Testing]**
  36. Where does the data come from to load the testing database? Is it real data from the customer? (Is the customer concerned about the security of this data?) **[Testing, Sys. Admin]**
  37. How is the training database accessed? Does the user input a special ID or phone number so the same resources can be shared on the system? **[Training, Database]**
  38. Is the training database separate from the live database? What is the procedure to reload it to a known state? **[Training, Database]**
  39. How often (or under what conditions) must the data be updated on a user's screen? **[User I/F]**
  40. The easiest way to define an interface to another system is to deliver some software, with libraries, and then have the other system meet that interface. **[User I/F]**

---

\* STREAMS is a trademark of UNIX System Laboratories, Inc.



- Performance**
1. Are there processes through which the application is single-threaded? What are the implications of this? **[Architecture]**
  2. How much headroom have you built into the architecture? **[Architecture, Modeling]**
  3. What is the system process structure? **[Architecture, Modeling]**
  4. Is a standard operating system being employed? If a variant of the operating system is being employed, how does it differ and why are the differences necessary? **[Architecture, Platform]**
  5. Can disk space be recovered while the system is active? **[Database]**
  6. Can your database be mapped onto the relational model? **[Database]**
  7. Does the database management system allow concurrency? If so, what's the granularity of locking (files vs. block vs. record)? **[Database]**
  8. How is data represented to the software [that is, how much "information hiding" is going on]? **[Database]**
  9. If applicable, what's the size of the application's database? **[Database]**
  10. Is the DBMS one that is supported under the current operating system/under UNIX® (for example, ORACLE\*, INFORMIX\*, etc.)? **[Database]**
  11. If you are planning to use your own Data Base Management System (DBMS), what are the development costs and trade-offs of using a commercial package, and what are the long-term support and maintenance costs? **[Database]**
  12. What are the characteristics of the database management system that you require? **[Database]**
  13. Will SQL be used to access the database? If not, why not? **[Database]**
  14. In a distributed system, what are the criteria for the distribution of data (geography, function, political, etc.)? **[Database, Dist. Sys.]**
  15. Have you determined where the data should reside—memory and/or disk? (Jim Gray's article on the "5 Minute Rule" for data is good metric to use.) **[Database, I/O, Modeling]**
  16. Does the file system require periodic reorganization to reclaim/compact the files for faster access? How is this monitored (to determine when it is to be done), and what is the effect on the system operations when it is done? **[Database, I/O, Sys. Admin]**
  17. How much file system space is consumed in a day/week of uninterrupted

---

\* ORACLE is a trademark of Oracle Corporation; INFORMIX is a trademark of Informix Corporation

operation? (Disk utilization for the database and/or UNIX® files may increase over time.) **[Database, Perf. Mon.]**

18. What is the performance impact of a fault tolerant system? Is it growable to meet future needs? **[Fault Tolerant]**
19. What processor or processors will be employed? **[Hardware]**
20. If new hardware is involved, what is the phasing to bridge it over to the existing system? **[Hardware, Migration]**
21. What do you do in a real-time system to reduce the disk fragmentation? **[I/O]**
22. How fast are the individual network links? **[I/O, Dist. Sys.]**
23. What communications hardware is employed on each processor and to interconnect processors? **[I/O, Dist. Sys.]**
24. Is there a standard methodology that the project is using for its development? **[Methodology]**
25. Will the optimizer on the compiler generate significantly better code? Has the entire system been compiled with this option? Are you sure it works? **[Methodology, Testing]**
26. How is the new system introduced? **[Migration]**
27. Is it possible to run with the “old” database and then convert later when the system is stable? (This might make it easy to “unconvert.”) **[Migration]**
28. Is the software run in parallel (on the same/different processor)? **[Migration]**
29. Does a performance model exist for the software architecture? **[Modeling]**
30. What tools are available to the user to size the system for a given configuration? **[Modeling]**
31. Is processing necessary to convert the representation into a form more appropriate for manipulation by application software (for example, does the database store ASCII representations of numbers or fields that must be parsed?) If so, what is the processing overhead? **[Modeling, Database, Architecture]**
32. What is the resource budget for each function/process in the system? **[Modeling, Perf. Mon]**
33. What are the benefits of an object-oriented programming approach? Is C++ being considered for the development phase? **[OOP]**
34. Are the measurement mechanism always active or only by request? If only by request can the mechanisms be turned on/off while the application is running? **[Perf. Mon.]**
35. Can you account for all the system calls that the program is making? Have you charted out the “normal” sequence of events? **[Perf. Mon.]**

36. Does the software incorporate mechanisms to monitor performance? **[Perf. Mon.]**
37. Does the software incorporate monitoring mechanisms that alarm when critical performance thresholds are exceeded? **[Perf. Mon]**
38. How many switches per second, swaps per second (page faults per second), inputs/outputs per second, messages per second, system calls per second is the system performing (by processor)? **[Perf. Mon]**
39. How much physical memory is left to handle processes that execute infrequently? **[Perf. Mon]**
40. Is there software in the system to dynamically measure the system performance and to output reports when exceptions occur? **[Perf. Mon]**
41. The resource budgets that are allocated to functions must take into account overload situations where the budgets will have to be adjusted to different values, that is, the budget will vary with the state of the system. How do functions detect this and make the adjustments? **[Perf. Mon]**
42. What are reasonable values for the amount of resources that functions/features in the system can consume? Have the responsible developers been informed of these limits? What mechanisms are in place to track them? **[Perf. Mon]**
43. What is the split between user time, system time, wait input/output time and idle for the maximum load that can be presented to the system? **[Perf. Mon]**
44. Characterize "normal" expected performance in terms of ratios of the metrics above, for example switches/swap, system calls/switch, etc. **[Perf. Mon., Architecture]**
45. Is there enough physical memory so that swaps or page faults are not induced for the active set (can the active set be locked in memory)? **[Perf. Mon., I/O]**
46. What resources are required to send or receive a message? **[Perf. Mon., I/O]**
47. How large is each process (text + data)? How large can it grow to be if it dynamically allocates memory **[Perf. Mon., Modeling]**
48. How much CPU and filesystem/database resource is consumed by the performance monitoring mechanisms? **[Perf Mon., Modeling]**
49. How much memory is required by the nominal set of active processes? How much shared memory can this nominal set of active processes consume? How much message memory can this nominal set consume? **[Perf. Mon., Modeling]**
50. In modeling the software (and in gathering performance data), can the user load be equated to an "equivalent unit of processing" (for example, knowing the cost of each transaction, the user can be told the capacity of

the system for a specific scenario)? [**Perf. Mon., modeling**]

51. Viewed individually, is the service rate per message adequate for the expected arrival rate? [**Perf. Mon., Modeling**]
52. What are the critical processing constraints—response time requirements, timing constraints between processes or external processors, throughput requirements, etc.? [**Perf. Mon., Modeling**]
53. Will the system collect data to indicate how close to exhaustion it is when it is running? [**Perf. Mon, Modeling**]
54. How is the system (network and processors) administered? Are performance-monitoring tools and mechanisms available through the hardware and controlling systems software to facilitate trouble isolation and resolution as well as performance tuning (load balancing)? [**Perf. Mon., Sys. Admin, Fault Isolation**]
55. What is the purpose of the prototype (test out assumptions, initial platform, gauge user impressions, etc.)? [**Prototype**]
56. What is the basic function of the application (that is, what problem is it trying to solve)? Is it a data collection system, a transaction processing system, etc.? [**Reqmnts., Architecture**]
57. Are the software modules being reused from another project? Are they being developed so that they can be reused by other projects? [**Reuse**]
58. Does your system include a fast global event recorder to capture hard-to-track interactions between modules? [**Testing, Perf. Mon., Fault Isolation**]
59. Does the application have unusual terminal interface requirements (for example, interactive, high-resolution graphics)? [**User I/F**]
60. How often (or under what conditions) must the data be updated on a user's screen? [**User I/F**]
61. Does the system require user programmability? How is this provided? Are there features in the system (fair share scheduler) to prevent a naive user from "locking up" the system with an unconstrained request? [**User Prog.**]