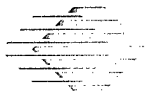


Carnegie Mellon
Software Engineering Institute

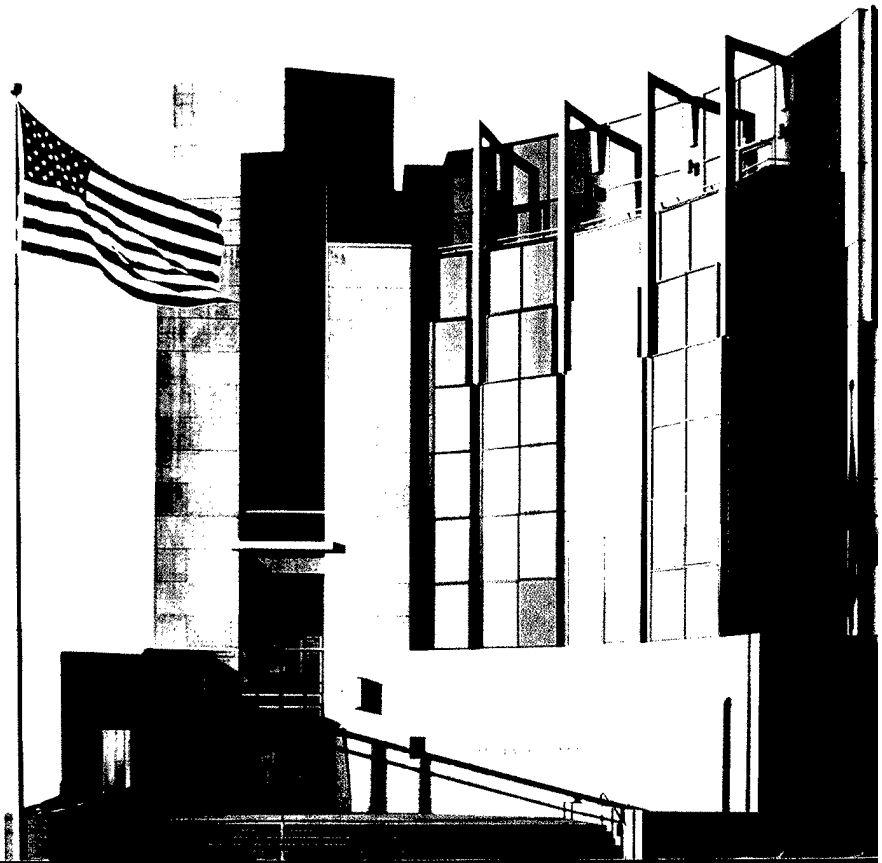
Active Reviews for Intermediate Designs

Paul C. Clements

August 2000

Architecture Tradeoff Analysis

Technical Note
CMU/SEI-2000-TN-009



Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state, or local laws or executive orders. However, in the judgment of the Carnegie Mellon Human Relations Commission, the Department of Defense policy of "Don't ask, don't tell, don't pursue" excludes openly gay, lesbian and bisexual students from receiving ROTC scholarships or serving in the military. Nevertheless, all ROTC classes at Carnegie Mellon University are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2056.

Obtain general information about Carnegie Mellon University by calling (412) 268-2000.

Active Reviews for Intermediate Designs

Paul C. Clements

August 2000

Architecture Tradeoff Analysis

Technical Note
CMU/SEI-2000-TN-009

Contents

List of Tables

Abstract

This paper introduces a technical review approach that is a blend of a stakeholder-centric, scenario-based, architecture evaluation method such as the Architecture Tradeoff Analysis Method[SM] (ATAM[SM]) [Kazman 00], and an *active design review* (ADR) of design specifications [Parnas 85]. There is a need for a technical review of a design finer-grained than an architecture, but not yet completely documented. Such a review exposes the design to its user community of application programmers, and allows early feedback on the overall approach, before the design is institutionalized in a detailed specification. This paper describes a recently piloted software design review technique that we call Active Review for Intermediate Designs (ARID). A hybrid of ADRs and scenario-based methods such as the ATAM, ARID fills a niche in the spectrum of technical design techniques between architecture at one end, and design specification documents at the other.

[SM] Architecture Tradeoff Analysis Method and ATAM are service marks of Carnegie Mellon University.

1 Introduction

As part of ongoing customer work, members of the SEI's Product Line Systems Program were recently asked to conduct a technical review of a design of a *portion* of a software system. This "portion" provided a cohesive, user-invocable service but did not correspond to any simple subset of the architecture: it spanned two of the architecture's layers, and included parts of several different work assignments.

It is said that when one's tool is a hammer, every problem looks like a nail. In cases like this one, our hammer tends to be the Architecture Tradeoff Analysis Method (ATAM), a robust and comprehensive method for evaluating a software architecture in the context of the quality-attribute goals that the architecture must meet [Kazman 00]. But the object of this review was not an architecture, and our hammer began looking suspiciously like a sledgehammer.

The designer who asked for our help wanted two things:

- First, he wanted to know if the design was tenable. That is, he wanted to know if writers of applications that use it could do so effectively. Did it provide the necessary services? Did it provide those services in a usable manner? Was the design conceptually coherent? Would the services impose too high a performance penalty?

- Second, he wanted to unveil the design to the community of software writers who needed it to do their jobs. Assuming that the basic design was sound, he wanted the design review to serve as the debut of the design to its consumers. In short, he wanted the participants to be trained, and he wanted their buy-in.

The design was not yet well documented. The invocable programs constituting the services were designed, meaning that they had been named, their parameters' types had been listed, and a rough statement of their semantics had been sketched. But the details that would have been provided by complete documentation were missing, such as:

- exceptions that might be raised by each program

- what would happen if the programs were called in illegal states

- how to declare variables of some of the types

- how programs in different processes, or on different processors, communicated and shared data with each other

We recognized that reviewing a design in its prerelease stages provides valuable early insight into the design's viability, and would allow for timely discovery of errors, inconsistencies, or inadequacies. Further, we recognized that most projects go through this preliminary stage for each and every one of their major components or subsystems. And yet none of the design review techniques in our repertoire seemed appropriate.

We began to attack the problem by considering two approaches: active design reviews, and scenario-based design review techniques.

1.1 Active Design Reviews

Active design reviews are an effective technique for ensuring quality, detailed designs in software [Parnas 85]. The method relies on actively engaging the reviewers by assigning them review tasks that are carefully structured to avoid asking yes/no questions. Such questions can undermine a review if a reviewer gives a carelessly considered answer. Rather, an active design review asks the reviewer to utilize the design in a series of exercises that test actual, not merely feigned, understanding.

For example, a conventional design review often begins with the chosen multitude of reviewers receiving stacks of read-ahead documentation, which they may or may not examine. The masses are then assembled in a central location on the appointed day. There is a presentation of the design being analyzed, and the design leader then opens the floor for discussion. A somewhat less benign review technique is to follow a checklist that asks reviewers to make sure that the design meets certain standards. But both of these are susceptible to reviewer boredom: even with the best of intentions, reviewers often find it easier to give designers the answer they desire, rather than the right one. And a high-pressure group setting will compel some reviewers to say nothing at all.

The following table illustrates some of the differences that one might observe between an active design review and a conventional review.

Table 1: Conventional vs. Active Design Review Questions/Instructions

Conventional Design Review Questions	Active Design Review Instructions
Are exceptions defined for every program?	Write down the exceptions that can occur for every program.
Are the right exceptions defined for every program?	Write down the range or set of legal values of each parameter. Write down the states under which it is illegal to invoke the program.
Are the data types defined?	For each data type, write down • an expression for a literal value of that data type; • a declaration statement to declare a variable for that type; • the greatest and least values in the range of that data type.
Are the programs sufficient?	Write a short pseudo-code program that uses the design to accomplish {some defined task}.
Is the performance of each program adequately specified?	For each program, write down its maximum execution time and list the shared resources that it may consume.

ADRs are primarily used to evaluate detailed designs of coherent units of software, such as modules or components. The questions tend to address (a) quality and completeness of the documentation; and (b) sufficiency, fitness, and suitability of the services provided by the design. The reviewers are chosen to represent consumers of both the design and the documentation that they must read to utilize it.

1.2 Scenario-Based Design Review Techniques

Scenarios are narrative descriptions of interactions of stakeholders with a system of interest. Scenarios have proven to be valuable in the evaluation of system and software designs, particular in the evaluation of software architectures [Abowd 96]. Exemplary of the scenario-based architecture evaluation methods, the Architecture Tradeoff Analysis Method is a proven evaluation method for software architectures, which are in some sense at the high end of the software design spectrum. Like ADRs, the ATAM also relies on the presence of consumers of the design, which it refers to as stakeholders. Stakeholders may be customers, users, testers, maintainers, performance engineers—anyone who has a vested interest in the sufficiency and fitness of the architecture. The ATAM proceeds by having the architect, and then the stakeholders, articulate the business drivers and then the specific quality and behavioral requirements that are incumbent upon the architecture. They do this by building a utility tree to describe the space of qualities of interest, and articulating each of the qualities by stating scenarios. A maintainer might describe a modification that may one day be needed, the performance engineer may give a benchmark that the architecture must satisfy, and so

forth. In the ATAM, the architecture is then analyzed by comparing the required qualities to known properties of approaches used in the architecture, and by using the scenarios as criteria against which to "test" (via analysis or thought experiment) how well the architecture fares.

Table 2 shows the steps of the ATAM.

Table 2: Steps of the ATAM

ATAM Steps	Description
Present the ATAM	Evaluation team leader presents a method overview to the participants.
Present business drivers	Client or representative of system whose architecture is being evaluated presents the business drivers underlying the architecture.
Present architecture	Architect makes presentation.
Identify architectural approaches	Evaluation team catalogs architectural approaches used, as basis for subsequent analysis
Generate quality attribute utility tree	Participants build utility tree to identify quality attributes (and the scenarios that express them) of interest. Evaluation team facilitates.
Analyze architectural approaches	Evaluation team and architect perform analysis based on qualities desired and approaches used.
Brainstorm and prioritize scenarios	The architecture's stakeholders adopt an additional set of scenarios expressing architectural requirements. Evaluation team facilitates.
Analyze architectural approaches	Evaluation team and architect perform analysis based on these new scenarios.
Present results	Evaluation team leader makes presentation of analysis results, lists identified risks, sensitivity points, and tradeoffs in the architecture.

2 ARID: An ADR/ATAM Hybrid

Active Design Reviews and the ATAM both bring useful features to bear on the problem of evaluating preliminary designs.

In an active design review, stakeholders receive detailed documentation and then complete exercise questionnaires on their own. But in the case of our design problem, there was no detailed documentation. Further, while ADRs eschew central meetings to achieve a higher-fidelity review result, one of our goals was to achieve group buy-in, and individual work would not accomplish that.

On the other hand, the ATAM is geared to evaluating a whole architecture, and not a portion of it. Further, the quality of interest in our case was limited to usability. Business drivers, utility trees, and elicitation and analysis of architectural approaches were all obviated.

Clearly something else was needed. We were adamant about not falling back on the standard technique of gathering the stakeholders in a room, presenting the design, and asking "What do you think?" And we recognized that ADRs and the ATAM both had strong qualities that could be brought to bear on this problem. An ADR requires active reviewer participation, which we thought would be ideal on two fronts. First, ADRs assure high-fidelity responses on the part of the reviewers: they can't simply sit there and say nothing. Second, active participation would, we hoped, increase the likelihood of group buy-in. From the ATAM, we embraced the idea of stakeholder-generated scenarios that would form the basis of the exercise. Rather than having the designers tell the services' consumers what usability meant, the consumers would tell the designers, thus setting the stage themselves for what it meant to pass the review. That, we reasoned, would also help with buy-in: if the reviewers set the conditions for the test, and the design passed the test, then surely that would signal that the design was usable.

Thus was born Active Reviews for Intermediate Designs, or ARID. By combining the best of active design reviews and scenario-based architecture evaluation methods such as the ATAM, ARID fills a niche in the spectrum of design review techniques.

2.1 ARID Participants

An ARID exercise requires three groups of participants:

- The ARID review team. Three roles must be filled. The facilitator will work with the lead designer to prepare for the review meeting, and then run the meeting itself. The scribe will capture the reviewers' inputs and results during the review meeting. And one

or more questioners are responsible for helping elicit and craft scenarios during the meeting. Optionally, a process observer can record where the exercise encountered difficulties and make suggestions for improvement to the method.

- The lead designer for the design being reviewed. This person is the spokesperson for the design, responsible for presenting it during the review, and the person to whom the result of the ARID will flow.

- Reviewers. Reviewers are drawn from the community of stakeholders for the design, people who have a vested interest in its adequacy and usability. They are the software engineers who will be expected to use the design and are the best people to judge its quality. So we merge the concept of ADR reviewers and ATAM stakeholders. As with the ATAM, we aim for approximately a dozen stakeholders, but this can vary depending on the size of the user community.

2.2 ARID Steps

An ARID exercise progresses across two phases that comprise nine steps. The table below summarizes:

Phase 1: Pre-meeting	Step 1: Identify reviewers
	Step 2: Prepare design presentation
	Step 3: Prepare seed scenarios
	Step 4: Prepare for the review meeting
Phase 2: Review meeting	Step 5: Present ARID method
	Step 6: Present design
	Step 7: Brainstorm and prioritize scenarios
	Step 8: Perform review
	Step 9: Present conclusions

Phase One of ARID is carried out as a meeting between the lead designer and the review facilitator. This meeting, in our case, lasted about a day. In this meeting, the following steps occur:

1. **Identify reviewers**. The lead designer and facilitator work together to identify the set of people who should be present at the review.

2. **Prepare design presentation.** The designer prepares a briefing explaining the design. In the case of our pilot application of ARID, this consisted of a two-hour viewgraph presentation that explained the services available, and then presented six examples in which the services were used together to solve different problems. The goal is to present the design in sufficient detail so that a knowledgeable audience member could use the design. Here, during Phase One, the designer should give a dry run of the presentation to the review facilitator, which is helpful for several reasons. First, it lets the facilitator see the design, and ask a set of "first order" questions that the reviewers would probably ask, thus helping the designer prepare. Second, it helps identify areas where the presentation could be improved. Third, it helps set the pace for the presentation itself, ensuring that the two-hour slot was not overrun. And fourth, it gives the designer practice in presenting the material to a critical audience.

3. **Prepare seed scenarios.** The designer and the review facilitator prepare a set of seed scenarios. Like the seed scenarios in the ATAM, these are designed to illustrate the concept of a scenario to the reviewers, who have an opportunity to see a sample set. The scenarios may or may not be used in the actual evaluation; the reviewers are free to adopt them or reject them in favor of others they brainstorm.

4. **Prepare for the review meeting**. Copies of the presentation, seed scenarios, and review agenda are produced for distribution to the reviewers during the main review meeting. The meeting is scheduled, reviewers are invited, and steps are taken to assure the presence of a quorum of reviewers at the meeting.

During Phase Two, the reviewers are assembled and the review meeting commences.

5. **Present ARID method.** The review facilitator spends 30 minutes explaining the steps of ARID to the participants.

6. **Present design.** The lead designer presents the two-hour overview presentation and walks through the examples. During this time, a ground rule is that no questions concerning implementation or rationale are allowed, nor are suggestions about alternate designs. The goal is to see if the design is usable, not to find out why things were done a certain way, or to learn about the implementation secrets behind the interfaces. Questions of factual clarification are allowed and encouraged. The facilitator enforces this rule during the presentation.

   During this time, the scribe captures each question, or each instance where the designer indicated that some sort of resource (usually a kind of documentation) was on its way but not yet available. The resulting list is summarized to show potential issues that the designer should address before the design could be considered complete and ready for production. In our case, the list of issues was captured on a whiteboard for all to see, and the facilitator made sure that all reviewers understood each issue and agreed with its wording before the presentation continued.

7. **Brainstorm and prioritize scenarios.** Just as in the ATAM, participants suggest scenarios for using the design to solve problems they expect to face. During brainstorming, all scenarios are deemed fair game. The seed scenarios are put into the pool with all the others. After a rich set of scenarios is gathered—in our case it was 20—a winnowing process occurs. Reviewers might suggest that two scenarios are versions of the same scenario, or that one subsumes another and should be merged. In our case, we eliminated five scenarios in this manner. After the pool of scenarios is winnowed, voting occurs. Each reviewer is allowed a vote total equal to 30% of the number of scenarios. They can allocate their votes to any scenario or scenarios they wish. The scenarios receiving the most votes will then be used to "test" the design for usability. After voting is complete, it is important to make the point that the reviewers have just defined what it means for the design to be usable: If it performs well under the adopted scenarios, then it must be agreed that the design has passed the review. Buy-in has begun.

8. **Perform review.** Beginning with the scenario that received the most votes, the facilitator asks the reviewers to jointly craft code (or pseudo-code) that uses the design services to solve the problem posed by the scenario. Reviewers make extensive use of the examples that were handed out as part of the designer's presentation. In our pilot case, the design was of a "virtual input/output" service suite whose purpose was to insulate application software from changes when, for example, a platform changed i/o channels or wiring harnesses or an application was ported to a different processor in the (distributed) system. The first scenario was this: "We move from a platform that uses two measured inputs to derive a third input, to a platform that includes a sensor to measure that third input directly." Code statements, captured by the scribe on a flipchart at the front of the room, were gathered that used the services to implement the first case. The group then captured what would have to change when the new platform was adopted. This first scenario, being the most popular (and thus, we hope, most important) was also among the most complex to solve; it took the group about three hours to finish it.

   During this time, a ground rule is that the designer is not allowed to help or give hints. In our exercise, the designer was tasked with "flying" the UML model on a direct-display projector, so that when participants had questions about a particular program's interface or interactions, he could quickly take us to the specification where the question could be answered.

   If, however, the group became stuck or began to go off in the wrong direction (which the designer indicated to the facilitator by pre-arranged signal), then the facilitator could stop the proceedings and ask the designer to get the group moving again by providing whatever information was deemed necessary. However, each time this happened, the facilitator asked the scribe to record as an issue where and why the group stalled, as this would indicate an area where the design (or the materials handed out to represent it) were insufficient to allow a non-expert to proceed. Any discrepancies uncovered during the review are also recorded as issues.

   This step continues until one of the following is achieved:

   a. The time allotted for the review has ended.

   b. The highest-priority scenarios have been exercised. Often the voting scheme produces a set of scenarios at the high end with several votes each, and a set of

scenarios at the low end, with one or zero votes each. When the former set is completed, it is usually safe to declare the review complete.

c. The group feels satisfied that a conclusion has been reached. Either the design is usable, which is indicated by the group quickly understanding how each subsequent scenario would be carried out as a straightforward variation on either the designer's examples or previously exercised scenarios, or the design is deemed unusable, which is indicated by the group finding some show-stopping deficiency. (In our case, the former occurred.)

9. **Present conclusions.** At the end, the list of issues is recounted, the participants are polled for their opinions regarding the efficacy of the review exercise, and they are thanked profusely for their participation.

3 Observations

Although we are only reporting on a single application of the approach, we will boldly hazard some informed speculation about its results.

3.1 Method Comparison

In comparing ARID to its conceptual ancestors, we focus on the following aspects:

	ATAM	ADR	ARID
Artifact examined	Architecture	Architectural and/or design documentation	Conceptual design approach, with embryonic documentation
Who participates	Architect, stakeholders	Experts in the design realm, consistency and completeness checkers	Lead designer, stakeholders
Basic approach	Elicit drivers and qualities, build utility tree, catalog approaches, perform approach-based analysis.	Construct review questionnaires, assign reviewers tasks, analyze their results to evaluate quality	Present design, elicit desired uses, have reviewers as a group use the design.
Outputs	Identified risks, sensitivity points, and tradeoff points	Errors, gaps, and inconsistencies	Issues and problems preventing successful usage
Approximate duration	3 full days of meetings, over approx. 2 weeks, plus unstructured work and communication between the meetings	1-2 days of reviewer work, 1 full day of reviewer de-briefing	1 day pre-meeting plus 1 day review meeting

3.2 Overall Results

We gathered a list of about two dozen issues, which confidentiality prevents us from sharing in detail. Only a few were about discrepancies in the design. Most of the remaining ones were process-oriented issues about using the design. These were technically outside the scope of the design's usability, but they were captured anyway. This served two purposes. First, it enabled the person who raised the issue to feel that he was heard, at which point we could move on. Second, it was believed that the organization at large would have to deal with those process issues eventually, and it provided the designer with a list of items that could be passed on as appropriate.

3.3 Participants' Reactions

A poll of the reviewers revealed that they were generally quite happy with the exercise. There was some initial concern that they might object to spending more than a day group-writing code, but in fact they seemed to enjoy the challenge of problem solving. By and large, they felt that the exercise was a good use of their time. Our "client," the software architect, was very pleased and felt that the exercise fully satisfied his goals of training, buy-in, and design review.

3.4 Duration

The all-hands part of the exercise consumed a day and a half. Twelve reviewers were present, plus the lead designer and two review team members (the facilitator and the scribe).

3.5 Possible Improvements

There were two major areas where we might have done better.

- By starting with the highest-priority scenario (the one receiving the most votes) we almost guaranteed that we would start with the most complicated one. In our case, working on this scenario took hours, and by the afternoon of the first day there was a glazed look in the participants' eyes. It might be better to start with a simpler scenario to keep people's energy high. With luck, the complex scenario might then be seen as a variant of a scenario that was handled previously, and can then be easily dispatched.

- Our scribe used a felt-tip pen and a flipchart to record the programming language statements dictated by the group. The group felt that a computer tied to a direct-display projector would have been better, because almost-alike statements could have been copied, pasted, and modified rather than written all over again. This would have saved time and reduced the tediousness of the exercise. Further, error corrections could be made much more cleanly.

- During the brainstorming step and the group-coding exercise, it is possible for a few participants to dominate the discussion and proceedings, while other reviewers sit idly by. This is not possible in a pure ADR, and is a weakness of ARID by comparison. It is incumbent upon the facilitator to make sure that all of the reviewers are engaged, even to

the point of shutting down the process and asking a quiet reviewer to name the next statement or propose the next scenario.

3.6 Observations

We can make the following observations about our experience with the ARID process:

- In the case of our pilot example, the designer was the chief software architect for the system at large. Our first experience showed that having architectural knowledge of the overall system was invaluable to have at hand, even if that knowledge is not embodied in the designer of the portion being reviewed. Questions came up about parts of the architecture that were not fully developed, and the reviewers needed answers in order to solve a problem at hand. In each case, the architect was able to help them make a reasonable assumption about the missing parts so that they could make progress. Even if the architect is not the lead designer of the portion being reviewed, the architect's presence is probably crucial to a smooth review.

- As in all reviews, the quality and expertise of the reviewers will determine the fidelity of the review results.

- There is a risk that design documentation that is promised will not be delivered. After all, the design has passed the review, and application programmers are using it to build products—so what's the point? That is dangerous software engineering, or perhaps more precisely, it is not software engineering at all but something less. The devil is often in the details, after all, and solving necessary details may sometimes precipitate changes in what was thought to be a stable design, even after the design is in the hands of the masses. That risk should be recognized and addressed by the organization conducting this kind of review.

4 Conclusions and Next Steps

The organization that sponsored the first ARID pilot has indicated that several more will likely occur because the organization is turning out its overall software architecture a portion at a time. This organization previously used both ADRs and the ATAM during the development of this architecture, and overall reaction to ARID has been positive. It was felt that ARID explored the design in more detail than did an ATAM, which is constrained by a need to allow time to focus on broad architectural issues and qualities other than the usability of a single part of the design. And ADRs, while poised to be the eventual litmus test for the promised documentation, were clearly not applicable in whole to the needs of this situation, which is certainly not unique in software development.

5 Acknowledgements

Our thanks go to Len Denbraber of Caterpillar, Inc., who helped conceive of the notion of a review somewhere between an ATAM and an ADR, and who was the client for the first ARID exercise. Lee clearly and effectively communicated the need, and pointed out why neither an ADR nor an ATAM seemed appropriate for the task at hand. ARID was born with his statement of need, and matured with his in-process and after-action suggestions. Thanks also go to Mark Klein and Rick Kazman, creative forces behind the ATAM, for their helpful comments and suggestions before we piloted this hybrid technique. Bob Krut did a yeoman job as ARID scribe, and suggested improvements. And thanks to Linda Northrop, director of the Product Line Systems Program at the SEI, for encouraging the trial and prodding this writeup.

References

[Abowd 96] Abowd, G.; Bass, L.; Clements, P.; Kazman, R.; Northrop, L.; and Zaremski, A. *Recommended Best Industrial Practice for Software Architecture Evaluation*, (CMU/SEI-96-TR-025). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. Available WWW <URL: http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.025.html> 1996.

[Kazman 00] Kazman, R.; Klein, M.; and Clements, P. *ATAM: Method for Architecture Evaluation*, (CMU/SEI-2000-TR-004). Pittsburgh, PA: Carnegie Mellon University. Available WWW <URL: http://www.sei.cmu.edu/publications/documents/00.reports/00tr004.html> 2000.

[Parnas 85] Parnas, D. and Weiss, D. "Active Design Reviews: Principles and Practice." *Proceedings, Eighth International Conference on Software Engineering,* 1985: 132-136.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (LEAVE BLANK)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 2000	Final

4. TITLE AND SUBTITLE
Active Reviews for Intermediate Designs

5. FUNDING NUMBERS
C — F19628-00-C-0003

6. AUTHOR(S)
Paul C. Clements

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-2000-TN-009

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12.A DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified/Unlimited, DTIC, NTIS

12.B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

This paper introduces a technical review approach that is a blend of a stakeholder-centric, scenario-based, architecture evaluation method such as the Architecture Tradeoff Analysis Method℠ (ATAM℠), and an active design review (ADR) of design specifications. There is a need for a technical review of a design that is finer-grained than an architecture, but not yet completely documented. Such a review exposes the design to its user community of application programmers, and allows for early feedback on the overall approach, before the design is institutionalized in a detailed specification. This paper describes a recently piloted software design review technique that we call Active Review for Intermediate Designs (ARID). A hybrid of ADRs and the ATAM, ARID fills a niche in the spectrum of technical design techniques between architecture at one end, and design specification documents at the other.

14. SUBJECT TERMS
active design review, Active Review for Intermediate Designs, ADR, architecture evaluation method, Architecture Tradeoff Analysis Method, ARID, ATAM, design specification, software architecture, software design, software documentation, technical review

15. NUMBER OF PAGES
20 pp.

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL