**Experiment No.: 10**

**Learning Objective:** Student should be able to understand Case Studies on following terms :-

1. Architecture evaluation, analysis and design
2. Architecture Tradeoff Analysis Method (ATAM)
3. Quality Attribute Workshops (QAW) 
4. Architecture reconstruction

**Theory:**

**1.Architecture Evaluation,analysis and design :**

Architecture evaluation is a development life-cycle activity whereby several stakeholders analyze the software architecture together in a formal or informal process using an assessment technique such as scenarios. Evaluations can utilize the same architecture assessment techniques described. In some methods, such as ATAM, the stakeholders generate the utility tree and the scenarios as part of the evaluation process. The characteristics of the development organization affect when and how evaluations are conducted. In small organizations that have a less mature engineering culture, a formal evaluation proceeding may not be appropriate. However, many of the principles and patterns of the evaluation techniques can still be applied. The architect may document a handful of critical scenarios that are used to guide the software design and that serve as input into the testing process.Formal evaluations should be done as soon as the architectural design concept is stable enough to be assessed, but before any real commitment to development has been taken. This allows the team to discover problems with the architecture at a time when such problems are easier and cheaper to address. However, evaluations *can* be performed after the system is under development or even when it is complete. In some cases I've had to reverse-analyze a system's architecture to determine why some undocumented design decisions were made when no one else on the team had any recollection of why. Although these were not formal evaluation proceedings, I used the same techniques of generating scenarios based on what stakeholders believed characterized the critical quality attributes and evaluated the existing system against them. Quite often the system that was built had quite different characteristics than what the stakeholders believed. In one case, the stakeholders believed that the system was flexible and could easily adapt to end-user information architecture and business process needs. The original designers, who were no longer on the project, had assured the other stakeholders that the system was flexible and adaptable. Because these qualities were not articulated specifically, there was no way to assess that the system satisfied the acquirers' concepts of flexible and adaptable. This situation is very common and is not a deliberate attempt to build something other than what the stakeholders are asking for. However, in the context of an *ad-hoc* development process and without proper specification of the system, acquirers are not able to articulate their specific needs and the developers are not capturing these specific quality requirements in any useful way. You can probably think of at least one such characteristic of the system that you have worked on. The acquirers want the user interface to have an acceptable response time or the system must be flexible enough to integrate into a customer's existing information architecture and Information Technology (IT) architecture. But these desires, if left at this level of specification, can rarely be satisfied completely or accurately. Remember, understand the right problem and solve the right problem. This is our imperative as software architects.

**There are several software architecture evaluation methods:**

**Scenario-based Architecture Analysis Method (SAAM).** This was probably the first documented software architecture analysis method and was originally developed to analyze an architecture for modifiability. However, it is useful for analyzing any nonfunctional aspect of an architecture. It is founded on the use of stakeholder-generated scenarios to assess an architecture.

**Architecture Trade-off Analysis Method (ATAM).** This is a successor of SAAM and is also gaining widespread use. This method incorporates quality attribute utility trees and quality attribute categories in the analysis of an architecture. Whereas SAAM does not explicitly address the interactions between quality attributes, ATAM does. Thus, the trade-offs are with respect to competing quality attributes. ATAM is a specialization of SAAM, specifically focusing on modifiability, performance, availability, and security.

**SAAM Founded on Complex Scenarios (SAAMCS).** This method considers the complexity of evaluation scenarios as the most important risk assessment factor.

**Extending SAAM by Integration in the Domain (ESAAMI).** This method integrates SAAM with domain-specific and reuse-based software development processes.

**Software Architecture Analysis Method for Evolution and Reusability (SAAMER).** This method focuses specifically on the quality attributes of evolution and reusability.

**Scenario-Based Architecture Reengineering (SBAR).** This method utilizes scenarios, simulation, mathematical modeling, and experience-based reasoning for assessing quality attributes. This method also incorporates an architecture design method.

**Architecture Level Prediction of Software Maintenance (ALPSM).** This is another method for analyzing maintainability using scenarios, called change scenarios, which represent maintenance tasks.

**Software Architecture Evaluation Model (SAEM).** This method is based on formal and rigorous quality requirements.

**Analysis Model**: Analysis model is the first step towards design. This model describes the structure of the system or application. It consists of class diagram and sequence diagrams that describes the logical implementation of the functional requirements. In this model, functionalities are modeled in terms of boundary, control and entity classes. Sequence diagrams realize use cases by describing the flow of events in the use cases when they are executed. These artifacts are categorized as High Level Design artifacts for assessment.

**Design Model:** Design model builds on analysis model by describing, in greater detail, the structure of the system and how the system will be implemented. A Design model consists of design classes structured into packages and subsystems with well-defined interfaces. This model describes the clear relationship between the design classes, and implementation elements. Implementation elements are directories and files including source code, data and executable files. These artifacts are analyzed for to the Detailed Design.

* **Implementation Model:** The Implementation Model represents the physical composition of the implementation in terms of Implementation Subsystems, and Implementation Elements (directories and files, including source code, data, and executable files).This artifact is considered in the assessment of the Detailed Design.
* **Deployment Model:** Deployment model describes the deployment units of a system and one or more physical network (hardware configurations on which these units are deployed. This artifact belongs to the Detailed Design part of an assignment.

**2. Architecture Tradeoff Analysis Method (ATAM)**

The Architecture Tradeoff Analysis Method (ATAM) is a method for evaluating software architectures relative to quality attribute goals. ATAM evaluations expose architectural risks that potentially inhibit the achievement of an organization's business goals. The ATAM gets its name because it not only reveals how well an architecture satisfies particular quality goals, but it also provides insight into how those quality goals interact with each other—how they trade off against each other.

The ATAM is the leading method in the area of software architecture evaluation. An evaluation using the ATAM typically takes three to four days and gathers together a trained evaluation team, architects, and representatives of the architecture's various stakeholders.

## **Challenges :**

Most complex software systems are required to be modifiable and have good performance. They may also need to be secure, interoperable, portable, and reliable. But for any particular system

* What precisely do these quality attributes such as modifiability, security, performance, and reliability mean?
* Can a system be analyzed to determine these desired qualities?
* How soon can such an analysis occur?
* How do you know if a software architecture for a system is suitable without having to build the system first?

Business drivers and the software architecture are elicited from project decision makers. These are refined into scenarios and the architectural decisions made in support of each one. Analysis of scenarios and decisions results in identification of risks, non-risks, sensitivity points, and tradeoff points in the architecture. Risks are synthesized into a set of risk themes, showing how each one threatens a business driver.

**The ATAM consists of nine steps:**

1. **Present the ATAM**. The evaluation leader describes the evaluation method to the assembled participants, tries to set their expectations, and answers questions they may have.
2. **Present business drivers**. A project spokesperson (ideally the project manager or system customer) describes what business goals are motivating the development effort and hence what will be the primary architectural drivers (e.g., high availability or time to market or high security).
3. **Present architecture**. The architect will describe the architecture, focusing on how it addresses the business drivers.
4. **Identify architectural approaches**. Architectural approaches are identified by the architect, but are not analyzed.
5. **Generate quality attribute utility tree**. The quality factors that comprise system "utility" (performance, availability, security, modifiability, usability, etc.) are elicited, specified down to the level of scenarios, annotated with stimuli and responses, and prioritized.
6. **Analyze architectural approaches**. Based on the high-priority factors identified in Step 5, the architectural approaches that address those factors are elicited and analyzed (for example, an architectural approach aimed at meeting performance goals will be subjected to a performance analysis). During this step, architectural risks, sensitivity points, and tradeoff points are identified.
7. **Brainstorm and prioritize scenarios**. A larger set of scenarios is elicited from the entire group of stakeholders. This set of scenarios is prioritized via a voting process involving the entire stakeholder group.
8. **Analyze architectural approaches**. This step reiterates the activities of Step 6, but using the highly ranked scenarios from Step 7. Those scenarios are considered to be test cases to confirm the analysis performed thus far. This analysis may uncover additional architectural approaches, risks, sensitivity points, and tradeoff points, which are then documented.
9. **Present results**. Based on the information collected in the ATAM (approaches, scenarios, attribute-specific questions, the utility tree, risks, non-risks, sensitivity points, tradeoffs), the ATAM team presents the findings to the assembled stakeholders.

The most important results are improved architectures. The output of an ATAM is an outbrief presentation and/or a written report that includes the major findings of the evaluation. These are typically

* a set of architectural approaches identified
* a "utility tree"—a hierarchic model of the driving architectural requirements
* the set of scenarios generated and the subset that were mapped onto the architecture
* a set of quality-attribute-specific questions that were applied to the architecture and the responses to these questions
* a set of identified risks
* a set of identified non-risks
* a synthesis of the risks into a set of risk themes that threaten to undermine the business goals for the system

## Benefits

* identified risks early in the life cycle
* increased communication among stakeholders
* clarified quality attribute requirements
* improved architecture documentation
* documented basis for architectural decisions

The most important results are improved architectures. The ATAM aids in eliciting sets of quality requirements along multiple dimensions, analyzing the effects of each requirement in isolation, and then understanding the interactions of these requirements.

## **Who Would Benefit**

Many people have a stake in a system's architecture, and all of them exert whatever influence they can on the architect to make sure that their goals are addressed. For example, the users want a system that is easy to use and has rich functionality. The maintenance organization wants a system that is easy to modify. The developing organization wants a system that is easy to build and that will employ the existing work force to good advantage. The customer wants the system to be built on time and within budget. All of these stakeholders will benefit from applying the ATAM. And needless to say, the architect is also a primary beneficiary.

1. **Quality Attribute Workshops (QAW) **

The QAW is a facilitated, early intervention method used to generate, prioritize, and refine quality attribute scenarios before the software architecture is completed. The QAW is focused on system-level concerns and specifically the role that software will play in the system. The QAW is dependent on the participation of system stakeholders—individuals on whom the system has significant impact, such as end users, installers, administrators, trainers, architects, acquirers, system and software engineers, and others. The QAW is an intense and demanding activity. It is very important that all participants stay focused, are on time, and limit side discussions throughout the day.

The QAW involves the following steps:

1. QAW Presentation and Introductions

2. Business/Mission Presentation

3. Architectural Plan Presentation

4. Identification of Architectural Drivers

5. Scenario Brainstorming

6. Scenario Consolidation

7. Scenario Prioritization

8. Scenario Refinement

The following sections describe each step of the QAW in detail.

**Step 1: QAW Presentation and Introductions**

In this step, QAW facilitators describe the motivation for the QAW and explain each step of the method. We recommend using a standard slide presentation that can be customized depending on the needs of the sponsor. Next, the facilitators introduce themselves and the stakeholders do likewise, briefly stating their background, their role in the organization, and their relationship to the system being built.

**Step 2: Business/Mission Presentation.**

After Step 1, a representative of the stakeholder community presents the business and/or mission drivers for the system. The term “business and/or mission drivers” is used carefully here. Some organizations are clearly motivated by business concerns such as profitability, while others, such as governmental organizations, are motivated by mission concerns and find profitability meaningless. The stakeholder representing the business and/or mission concerns (typically a manager or management representative) spends about one hour presenting

• the system’s business/mission context

• high-level functional requirements, constraints, and quality attribute requirements During the presentation, the facilitators listen carefully and capture any relevant information that may shed light on the quality attribute drivers. The quality attributes that will be refined in later steps will be derived largely from the business/mission needs presented in this step.

**Step 3:** **Architectural Plan Presentation**

While a detailed system architecture might not exist, it is possible that high-level system descriptions, context drawings, or other artifacts have been created that describe some of the system’s technical details. At this point in the workshop, a technical stakeholder will present the system architectural plans as they stand with respect to these early documents. Information in this presentation may include

• plans and strategies for how key business/mission requirements will be satisfied

• key technical requirements and constraints—such as mandated operating systems, hardware, middleware, and standards—that will drive architectural decisions

• presentation of existing context diagrams, high-level system diagrams, and other written descriptions.

**Step 4: Identification of Architectural Drivers**

In steps 2 and 3, the facilitators capture information regarding architectural drivers that are key to realizing quality attribute goals in the system. These drivers often include high-level requirements, business/mission concerns, goals and objectives, and various quality attributes. Before undertaking this step, the facilitators should excuse the group for a 15-minute break, during which they will caucus to compare and consolidate notes taken during steps 2 and 3. When the stakeholders reconvene, the facilitators will share their list of key architectural drivers and ask the stakeholders for clarifications, additions, deletions, and corrections. The idea is to reach a consensus on a distilled list of architectural drivers that include high-level requirements, business drivers, constraints, and quality attributes. The final list of architectural drivers will help focus the stakeholders during scenario brainstorming to ensure that these concerns are represented by the scenarios collected.

**Step 5: Scenario Brainstorming**

After the architectural drivers have been identified, the facilitators initiate the brainstorming process in which stakeholders generate scenarios. The facilitators review the parts of a good scenario (stimulus, environment, and response) and ensure that each scenario is well formed during the workshop. Each stakeholder expresses a scenario representing his or her concerns with respect to the system in round-robin fashion. During a nominal QAW, at least two round-robin passes are made so that each stakeholder can contribute at least two scenarios. The facilitators ensure that at least one representative scenario exists for each architectural driver listed in Step 4.

Scenario generation is a key step in the QAW method and must be carried out with care. We suggest the following guidance to help QAW facilitators during this step:

1. Facilitators should help stakeholders create well-formed scenarios. It is tempting for stakeholders to recite requirements such as “The system shall produce reports for users.” While this is an important requirement, facilitators need to ensure that the quality attribute aspects of this requirement are explored further. For example, the following scenario sheds more light on the performance aspect of this requirement: “A remote user requests a database report via the Web during peak usage and receives the report within five seconds.” Note that the initial requirement hasn’t been lost, but the scenario further explores the performance aspect of this requirement. Facilitators should note that quality attribute names by themselves are not enough. Rather than say “the system shall be modifiable,” the scenario should describe what it means to be modifiable by providing a specific example of a modification to the system.
2. The vocabulary used to describe quality attributes varies widely. Heated debates often revolve around to which quality attribute a particular system property belongs. It doesn’t matter what we call a particular quality attribute, as long as there’s a scenario that describes what it means.
3. Facilitators need to remember that there are three general types of scenarios and to ensure that each type is covered during the QAW:
4. use case scenarios - involving anticipated uses of the system
5. growth scenarios - involving anticipated changes to the system.

c. exploratory scenarios - involving unanticipated stresses to the system that can include uses and/or changes

4. Facilitators should refer to the list of architectural drivers generated in Step 4 from time to time during scenario brainstorming to ensure that representative scenarios exist for each one.

**Step 6: Scenario Consolidation**

After the scenario brainstorming, similar scenarios are consolidated when reasonable. To do that, facilitators ask stakeholders to identify those scenarios that are very similar in content. Scenarios that are similar are merged, as long as the people who proposed them agree and feels that their scenarios will not be diluted in the process. Consolidation is an important step because it helps to prevent a “dilution” of votes during the prioritization of scenarios (Step 7). Such a dilution occurs when stakeholders split their votes between two very similar scenarios. As a result, neither scenario rises to importance and is therefore never refined (Step 8). However, if the two scenarios are similar enough to be merged into one, the votes might be concentrated, and the merged scenario may then rise to the appropriate level of importance and be refined further. Facilitators should make every attempt to reach a majority consensus with the stakeholders before merging scenarios. Though stakeholders may be tempted to merge scenarios with abandon, they should not do so. In actuality, very few scenarios are merged.

**Step 7: Scenario Prioritization**

Prioritization of the scenarios is accomplished by allocating each stakeholder a number of votes equal to 30% of the total number of scenarios generated after consolidation. The actual number of votes allocated to stakeholders is rounded to an even number of votes at the discretion of the facilitators. For example, if 30 scenarios were generated, each stakeholder gets 30 x 0.3, or 9, votes rounded up to 10. Voting is done in round-robin fashion, in two passes. During CMU/SEI-2003-TR-016 11 each pass, stakeholders allocate half of their votes. Stakeholders can allocate any number of their votes to any scenario or combination of scenarios. The votes are counted, and the scenarios are prioritized accordingly.

**Step 8: Scenario Refinement**

After the prioritization, depending on the amount of time remaining, the top four or five scenarios are refined in more detail. Facilitators further elaborate each one, documenting the following:

• Further clarify the scenario by clearly describing the following six things:

1. stimulus - the condition that affects the system

2. response - the activity that results from the stimulus

3. source of stimulus - the entity that generated the stimulus

4. environment - the condition under which the stimulus occurred

5. artifact stimulated - the artifact that was stimulated

6. response measure - the measure by which the system’s response will be evaluated

• Describe the business/mission goals that are affected by the scenario.

• Describe the relevant quality attributes associated with the scenario.

• Allow the stakeholders to pose questions and raise any issues regarding the scenario. Such questions should concentrate on the quality attribute aspects of the scenario and any concerns that the stakeholders might have in achieving the response called for in the scenario.

**4 QAW Benefits :**

The QAW provides a forum for a wide variety of stakeholders to gather in one room at one time very early in the development process. It is often the first time such a meeting takes place and generally leads to the identification of conflicting assumptions about system requirements. In addition to clarifying quality attribute requirements, the QAW provides increased stakeholder communication, an informed basis for architectural decisions, improved architectural documentation, and support for analysis and testing throughout the life of the system.

The results of a QAW include

• a list of architectural drivers

• the raw scenarios

• the prioritized list of raw scenarios

• the refined scenarios

**This information can be used to :**

• update the organization’s architectural vision

• refine system and software requirements

• guide the development of prototypes

• exercise simulations

• understand and clarify the system’s architectural drivers

• influence the order in which the architecture is developed

• describe the operation of a system.

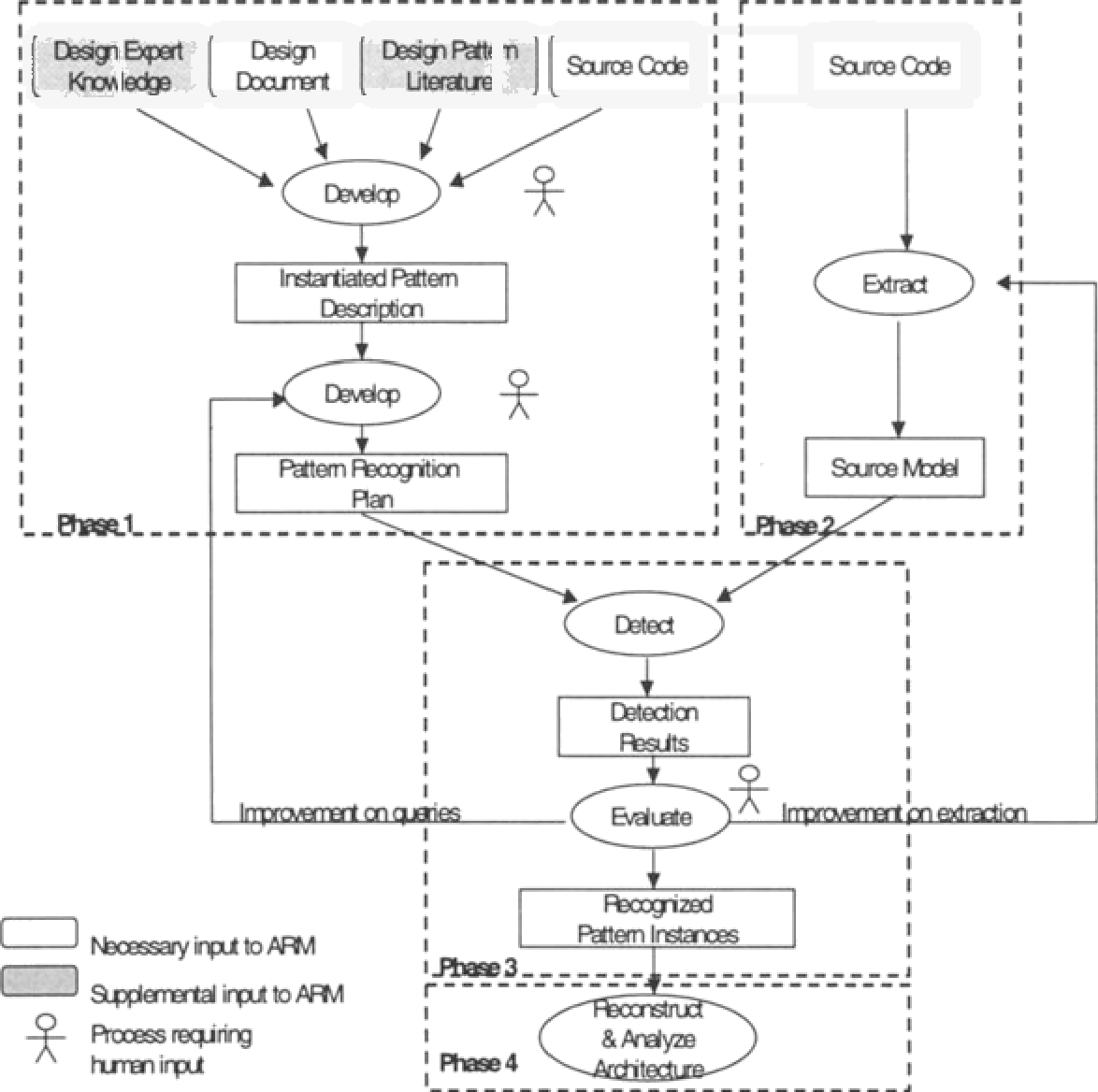
## **ARCHITECTURE RECONSTRUCTION METHOD**

To assist software architecture recovery of systems designed and developed with patterns, we developed the Architecture Reconstruction Method (ARM)—a semi-automatic analysis method for reconstructing architectures based on the recognition of architecmral patterns.

ARM is depicted in Figure 2. As indicated by the dashed boxes in this

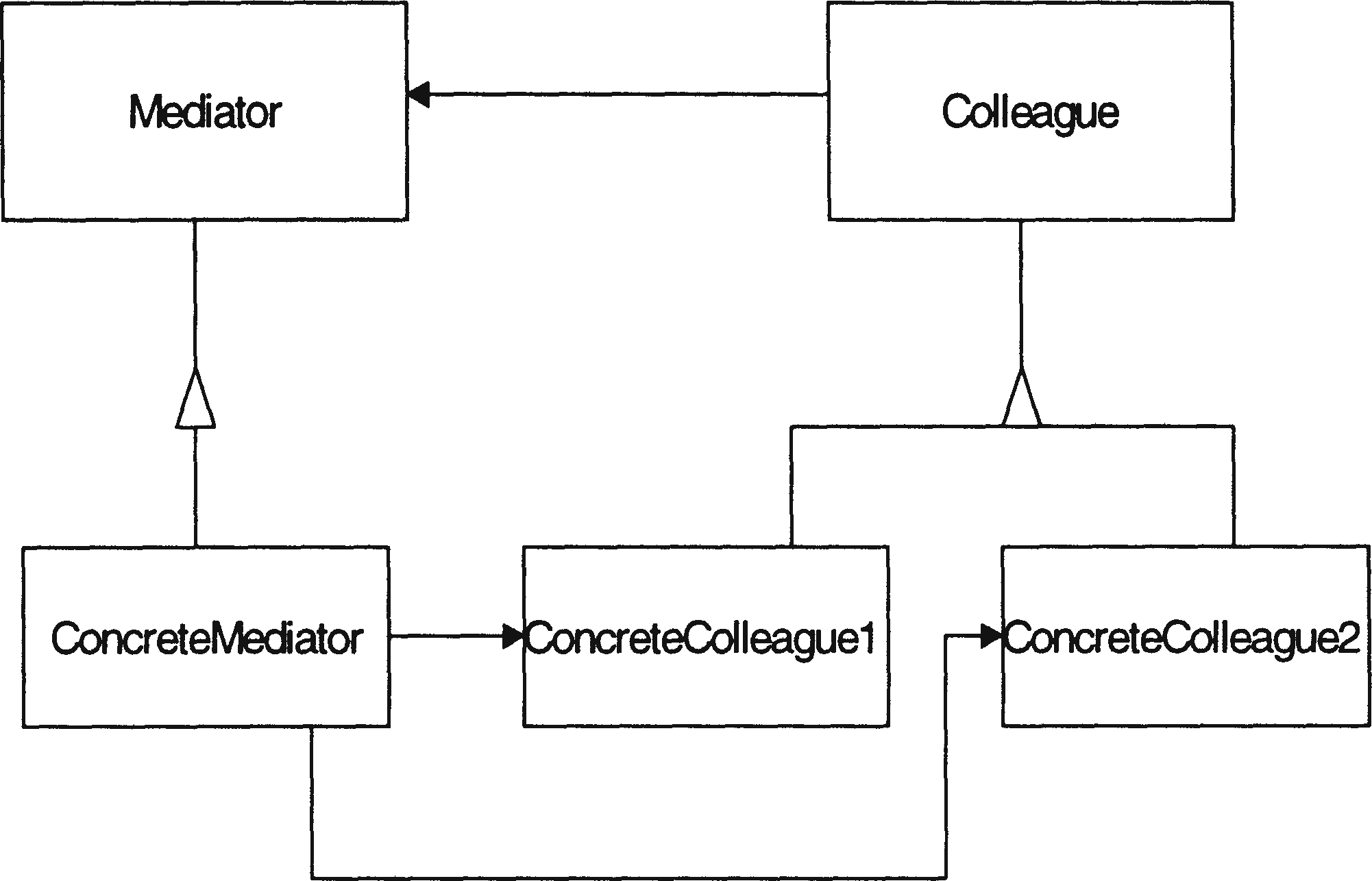
figure, ARM consists of four major phases:

1. Developing a concrete pattern recognition plan.
2. Extracting a source model.
3. Detecting and evaluating pattern instances.
4. Reconstructing and Analyzing the architecture.



Constructing a pattern recognition plan consists of three steps. The first is to develop an instantiated pattern description. By instantiation*,* we mean a concrete pattern description, with all the pattern elements and their relations described in terms of the constructs available from the chosen implementation language. Starting with a design document, one can manually determine the patterns used in the design and can extract the *abstract pattern rules—the* design rules that define a pattern’s structural and behavioral properties. Pattern descriptions found in the design pattern literature, e.g., Buschmann, et al., 1996, or obtained from humans who are familiar with the system design can be used to supplement these rules. Using these abstract pattern rules as a guide, one can then examine the source code of several potential pattern instances to derive the corresponding *concrete pattern rules—the* implementation rules that realize abstract pattern rules using data structures, coding conventions, coding methods and algorithms. Such concrete pattern rules can be recognized via syntactic cues, such as naming conventions and programming language keywords, or an analysis of data access and control flow. An instantiated pattern description is a specification of the concrete pattern rules written in Rigi Standard Format (RSF) (Wong, et al., 1994). A clause in RSF is a tuple (relation, entity1, entity2), which represents the relationship *entityl relates to entity2.* For example in the *Mediator design pattern* (see Figure 3), a *Mediator* component serves as the communication hub for all the *Colleague* components. An abstract pattern rule for this pattern is

“The Mediator component mediates communications between colleague components.”



*Figure 3:* Mediator design pattern

k one of our case studies, the *Mediator pattern* is implemented in a C++ class where mediator and colleague components are member functions.

## **Extracting a source model**

The second phase of ARM is to extract a source model that represents a system’s source elements and the relations between them. The output of this phase is a source model that contains the information that is used for detecting *necessary* pattern rules. For example, Table 1 shows some of the relations that Dali currently extracts from C++ programs (Kazman & Carriere, 1999). The relations needed for detecting the necessary pattern rules of the Presentation-Abstraction-Control (PAC) pattem2 (Buschmann, et al., 1996) in our case studies are denoted by \*

*Table 1:* Typical set of source relations extracted by Dali.

|  |  |  |
| --- | --- | --- |
| Relation | From | To |
| calls \* | function | function |
| contains | file | function |
| defines | file | class |
| has\_subclass \* | class | class |
| has\_friend | class | class |
| defines\_fn \* | class | function |
| has\_member \* | class | variable |
| defines\_var \* | function | variable |
| has\_instance \* | class | variable |
| defines global \* | file | variable |

var\_access \* function variable

A complication is that patterns are revealed at different levels of abstraction (e.g., the function vs. the class level), thus different pa8s of the recognition plan may need to be applied to a source model at different levels of abstraction. Using abstraction techniques, such as the *aggregation* technique provided by Dali (Kazman & Carriere, 1999), lower level source model elements can be grouped into a higher level element without loss of information. Thus one can use it to bring the source model to appropriate levels of abstraction for pattern detection and architecture analysis.

## **Detecting and evaluating pattern instances**

Detecting pattern instances using Dali is an automatic process in which one uses query tools to execute a recognition plan with respect to a source model. After running the recognition plan on the source model using the query tools, the detection output consists of all the pattern instance candidates. Human evaluation of these candidates is required to compare them with the designed pattern instances and determine which candidates are intended, which are false positives and false negatives. A false positive is a candidate which is not designed as a pattern instance, but is “detected” falsely as an instance. A false negative is a candidate which is designed as an instance, but is not detected as one.

One can try to improve the results (i.e., remove false positives and negatives) by modifying either the recognition plan or the source model and reiterating through ARM method. To improve the pattern recognition plan, one may choose another component of the pattern as the anchor and reorder the queries to form a new plan, or refine the query constraints for some of the pattern elements. If the source model extraction caused the deficiencies, an analyst needs to try to improve the extraction process by refining the existing extraction tools to catch the defects and/or incorporating other extraction tools to enhance the accuracy of source model, as described in (Kazman & Carriere, 1998).

However, if the source code is incomplete or if the pattern is defined by complex dynamic attributes, it may be impossible for the recognition technique to precisely detect all pattern instances. The evaluation process ends when Dali can detect the maximal set of true pattern instances, and the human analyst can explain the presence of false positive and the absence of false negative instances. The output is the set of validated pattern instances.

## **Reconstructing and analyzing the architecture**

### In the final step, the analyst uses a visualization tool, such as Rigi, to align the recognized architectural pattern instances with the designed pattern instances, organizing the other elements in the source model around the detected instances. The resultant architecture can be analyzed for deviations from the designed architectur

**Result and Discussion:**

* Architecture evaluation, analysis and design was conducted successfully.
* Architecture Tradeoff Analysis Method (ATAM) was understood
* Quality Attribute Workshops (QAW) was understood
* Architecture reconstruction was understood successfully

**Learning Outcomes:** Students should have the ability to

LO1: Define Architecture evaluation.

LO2: Identify Architecture Tradeoff Analysis Method.

LO3: Learn Quality Attribute Workshops.

LO4: Learn Architecture reconstruction.

**Course Outcomes:** Upon completion of the course students will be able to learn

1.Architecture evaluation, analysis and design.

2.Architecture Tradeoff Analysis Method (ATAM).

3.Quality Attribute Workshops (QAW). 

4.Architecture reconstruction.

**Conclusion:** Case study on the given topics were completed successfully.

For Faculty Use

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Correction Parameters** | **Formative Assessment [40%]** | **Timely completion of Practical [ 40%]** | **Attendance / Learning Attitude [20%]** |  |
|  |  |  |