A Survey on Software Architecture Analysis Methods

Liliana Bobrica and Eila Niemela

IEEE TOSE July 02

Group 1 and 6

Software Architecture Analysis Methods

CIS 740

Instructor: **Dr. David A. Gustafson**

Presented By

1. **Vikranth Vaddi**
2. **Hong Zhang**
3. **Sudarshan Kodwani**
4. **Travis Stude**
5. **Sandeep Pujar**
6. **Abhinav Pradhan**
7. **Srinivas Kolluri**
8. **Kiran Devaram**
9. **Saravana Kumar**

Why focus on Architecture…..!

purpose & goals

Software Architecture definition:

"A high level configuration of system components and the connections that coordinate component activities"

- ***Architecture is often the first artifact that represents decisions on how requirements of all types are to be achieved.As the manifestation of early design decisions that are hardest to change.***
- ***SAAM's goals is to verify basic architectural assumptions and principles against the documents describing the desired properties of any application.***
- ***SAAM (software) permits the comparison of architectures within the context of any organization's particular needs.***
- ***The purpose of SAAM is not to criticize or commend particular architectures, but to provide a method for determining which architecture supports an organization's needs.***
- ***A good understanding of system design at the architectural level makes it easier to detect design errors early on and easier to modify the system later.***

Background

- ***SAAM appeared in 1993, corresponding with the trend for a better understanding of general architectural concepts, as a foundation for proof that a software system meets more than just functional requirements.***
- ***Establish a method for describing and analyzing software architectures.***
- ***SAAM was initially developed for application early in design, it is validated in an analysis of several existing industrial systems.***

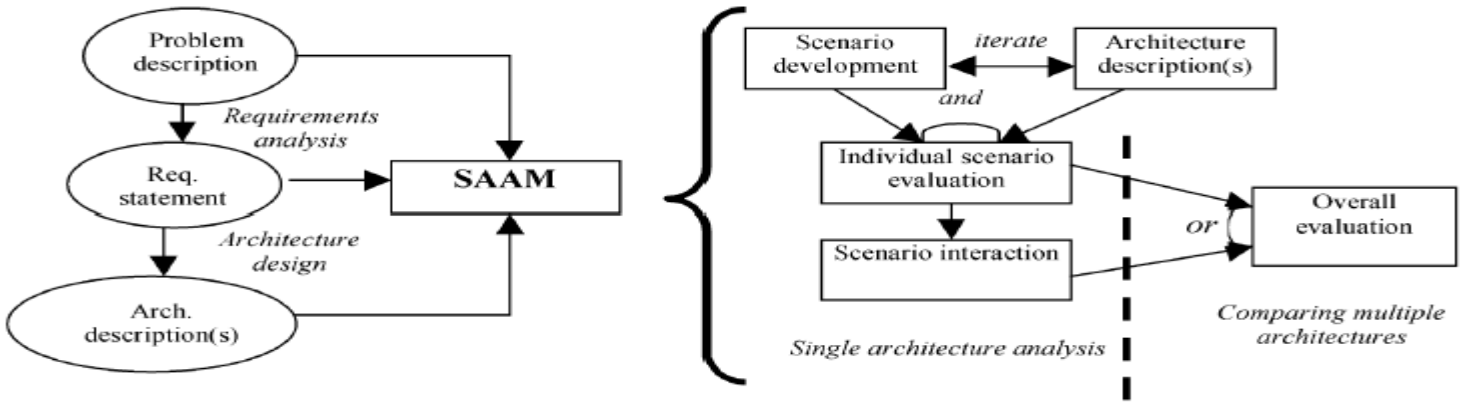


Fig. 1. SAAM inputs and activities.

Evaluating Architectures based on Software Quality

- Evaluating Architectures is difficult
- Motivation : Software Quality factors
  - Maintainability
  - Portability
  - Modularity
  - Reusability
- Perspective :
  - Functionality
  - Structure – Lexicon describing structure
  - Allocation

Main Activities

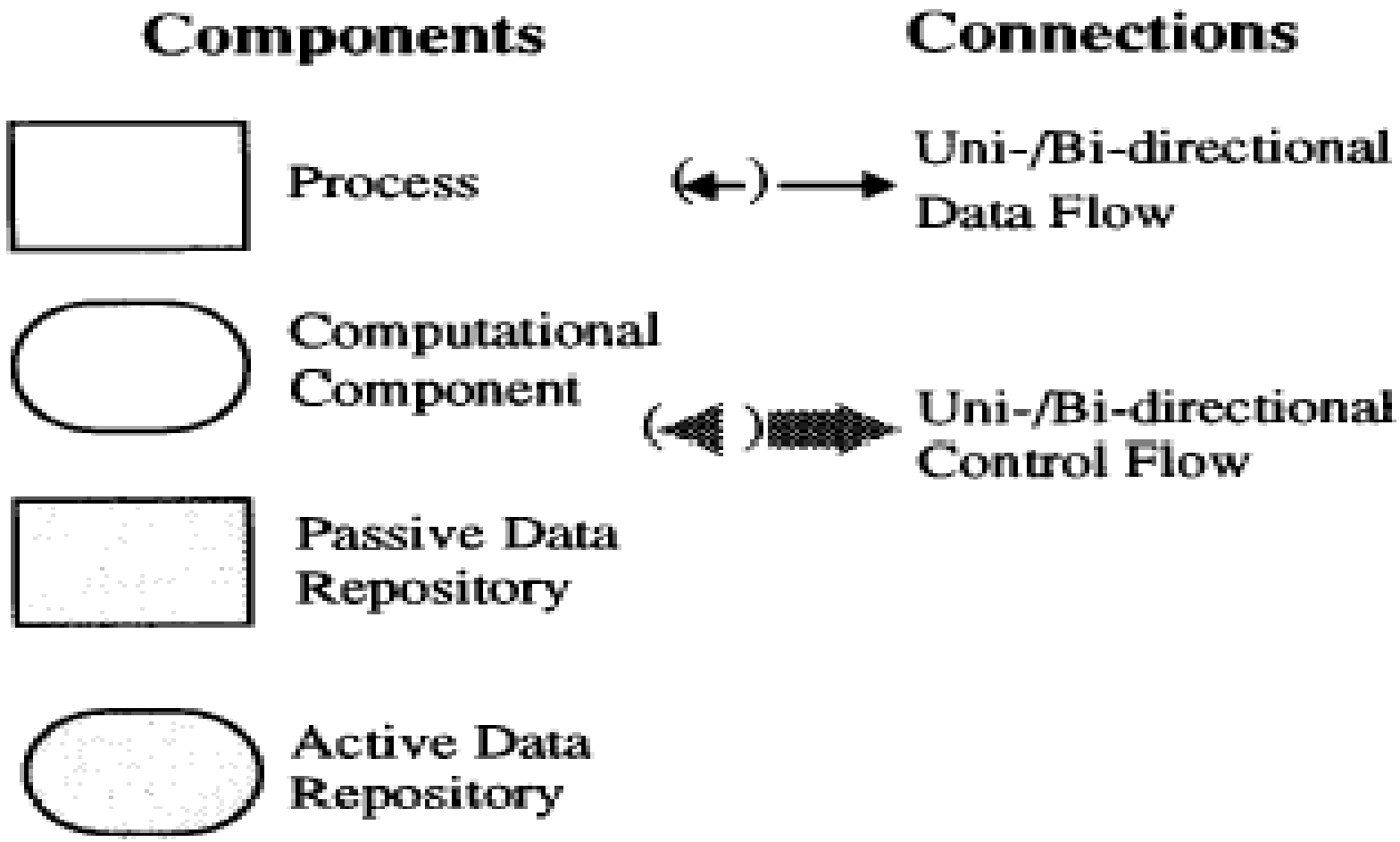
- Characterize a canonical functional partitioning for the domain
- Map the functional partitioning onto the architecture's structural decomposition
- Choose a set of quality attributes with which to assess the architecture
- Choose a set of concrete tasks which test the desired quality attributes
- Evaluate the degree to which each architecture provides support for each task.

Perspectives

- Functionality: What the system does
- Structure: How a system is constructed from smaller pieces
  - Components - represent computational entities
  - Connections – connections between components
- Allocation:

  - how intended functionality is achieved by the developed system.
  - differentiates architectures within a given domain

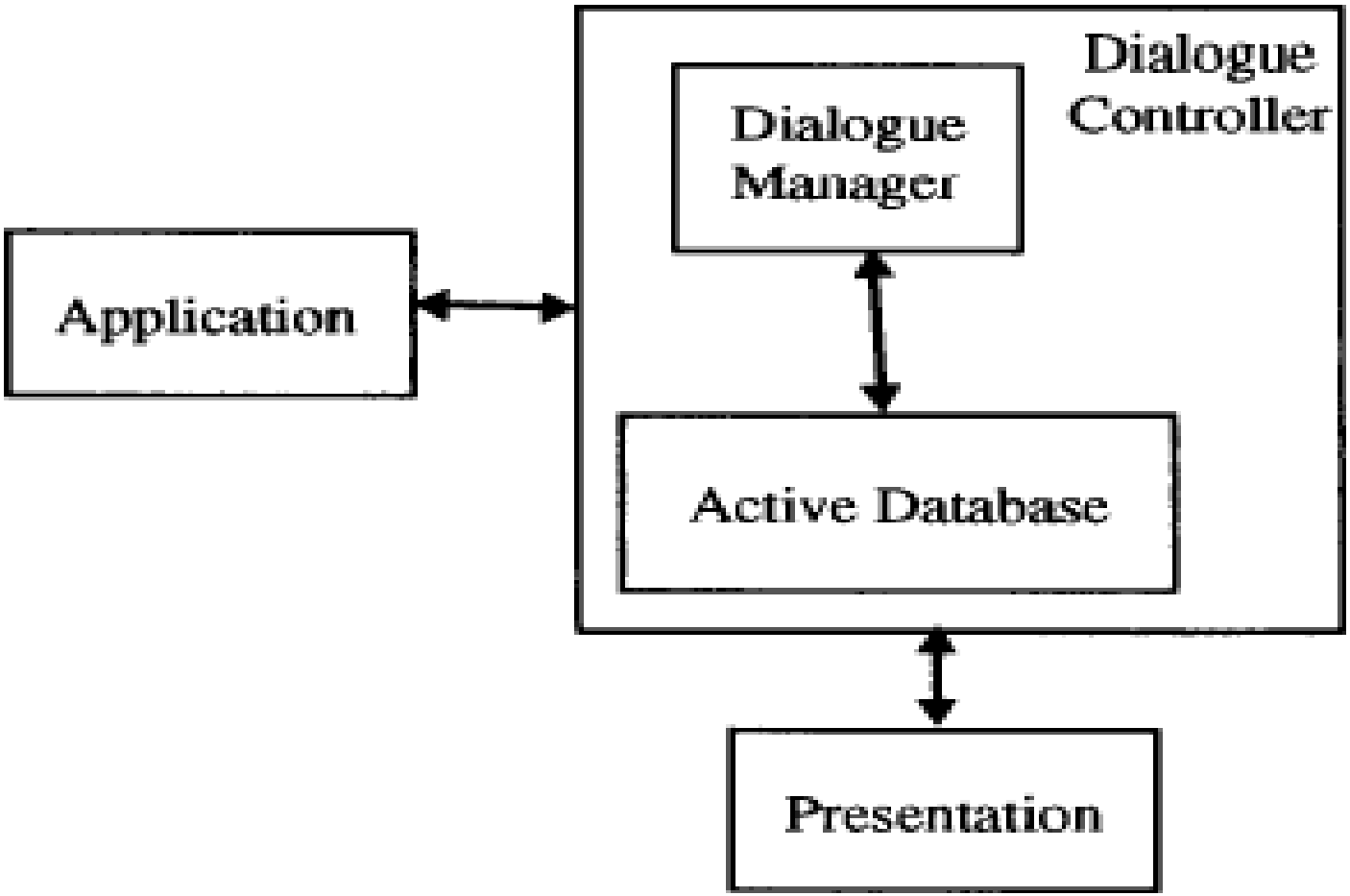
Lexicon for describing structure



Canonical functional partitioning
The Arch/Slinky metamodel

- Five basic functions of user interface software
  - Functional Core (FC)
  - Functional Core Adapter (FCA)
  - Dialogue (D)
  - Logical Interaction (LI) component
  - Physical Interaction (PI) component

Serpent


Dialogue Controller
Dialogue Manager
Application
Active Database
Presentation

Analyzing Architectural Qualities

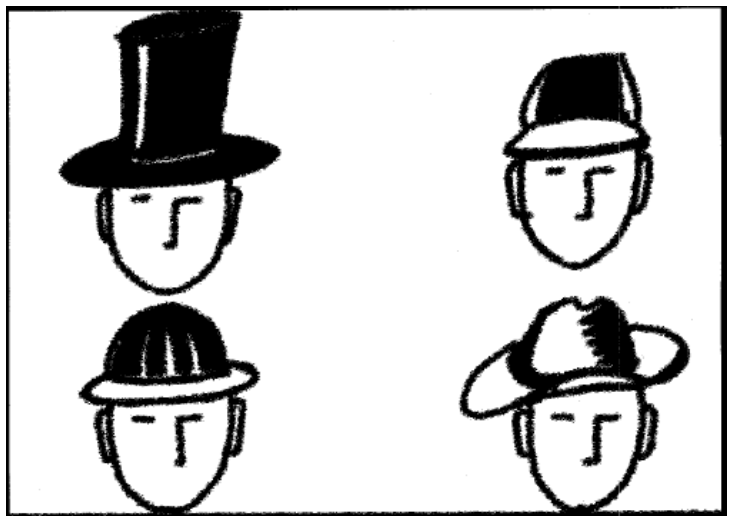
- Evaluate User Interface Architecture with respect to modifiability

- Example Modifications
  - Adaptation to new operating environments
  - Extension of capabilities

Analysis of Candidate system

- Changing the toolkit
  - Modification to the dialogue manager
  - No architectural support
- Adding a menu option
  - Easier to isolate because view controllers subdivide Dialogue Manager
  - Therefore Architectural support Provided

Scenario-Based Analysis Of Architecture

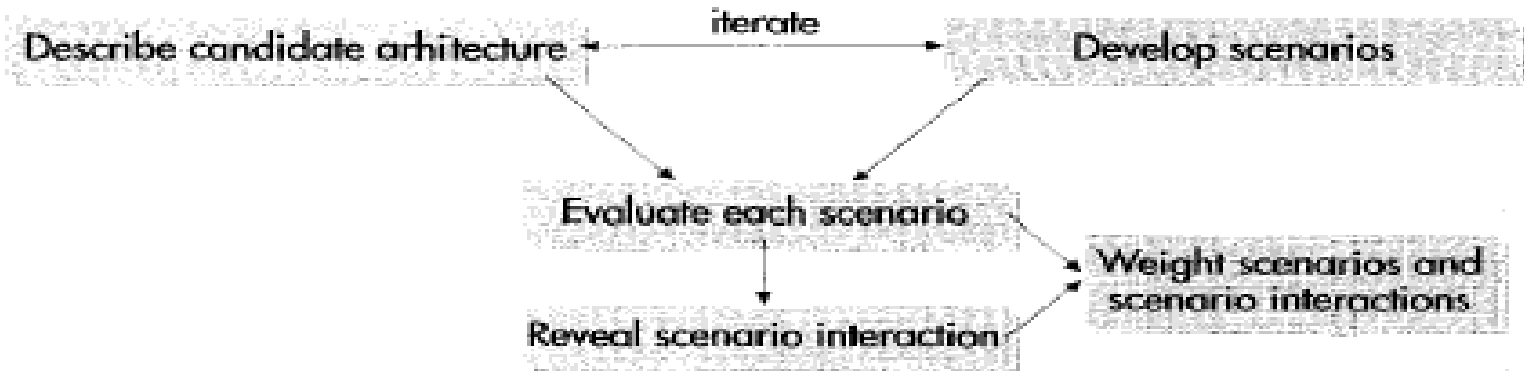


Why Scenarios..?

Scenarios Because……!

- ***Scenarios offer a way to review the vague quality attributes (modifiability security, safety or portability) in more specific circumstances by capturing system use contexts.***
- ***Developers can analyze the architecture with respect to how well or how easily it satisfies the constraints imposed by each scenario.***
- ***Scenarios help visualize the candidate architecture with various perspectives***
  - ***System operator***
  - ***System designer and modifier***
  - ***System administrator***
- ***Scenarios force designers to consider the future uses of, and changes to the system.***
- ***Designers would have to think on the lines of how the architecture will accommodate a particular change in the system and not what degree a system can be modified.***

Scenario Based - SAAM Methodology

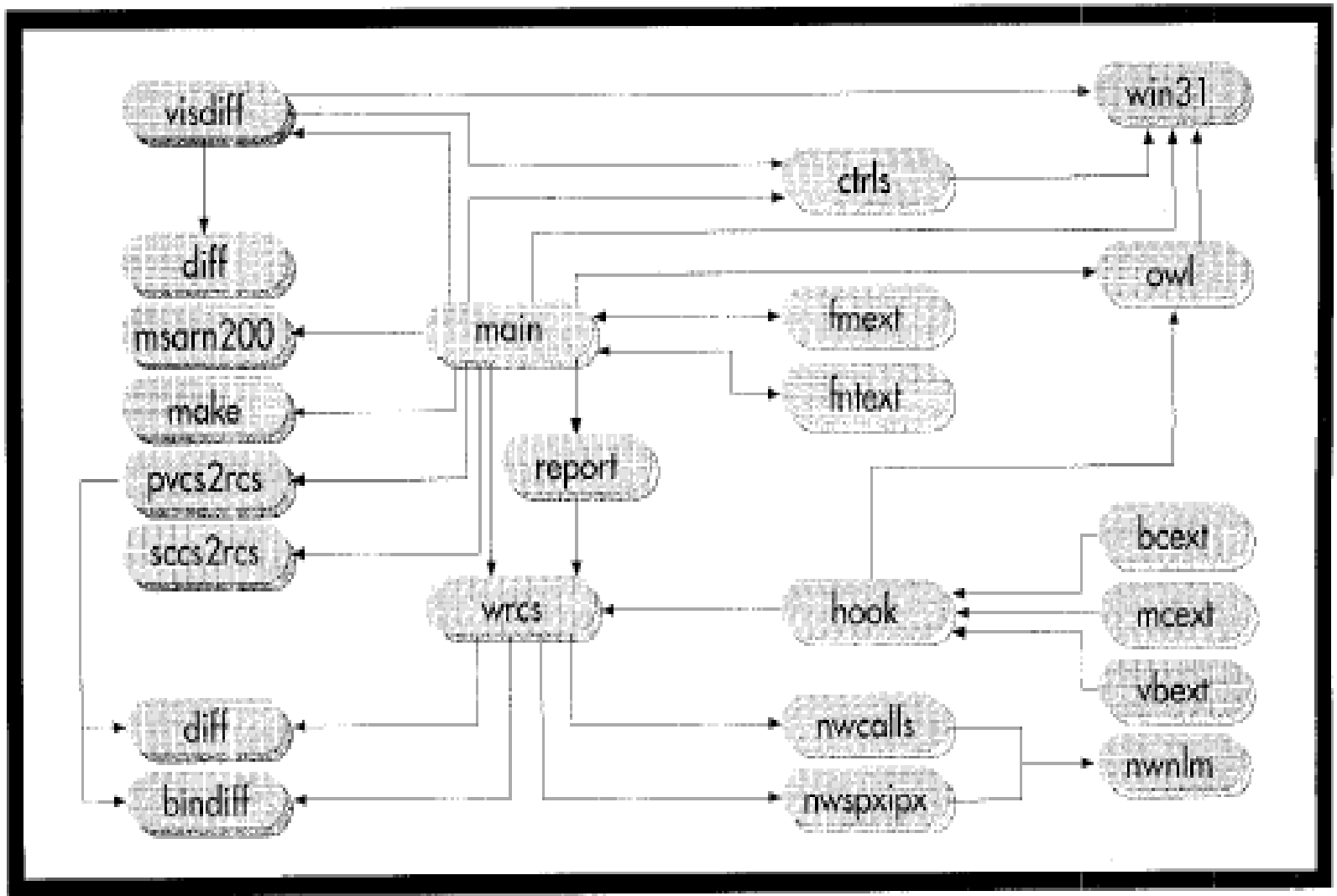


- ***Describe the candidate architecture***
- ***Develop Scenarios***
- ***Evaluate each scenario***
- ***Reveal scenario interaction***
- ***Weight scenarios & Scenario interaction***

Scenario Based - SAAM Methodology

- ***Describe the candidate architecture:***
- ***Develop Scenarios:***
- ***Evaluate each scenario:*** ***Each scenario should be classified into direct or indirect scenario.***
  - ***Direct Scenario: no architectural changes required.***
  - ***Indirect Scenario: architectural changes required.***
- ***Reveal scenario interaction: Weight scenarios & Scenario interaction:***

Architecture -WRCS



Steps in Analysis

Describe the candidate architecture:

Figure 3 shows the architecture

- *Develop Scenarios:*

For the user role, scenario included

- *Compare binary file representations*
- *Configure the toolbar*
- *Manage multiple projects*

- **For the Maintainer role, scenarios include**
  - *Port to another OS*
  - *Modify the user interface in minor ways*
- **For the administrator role, scenarios included**
  - *Change access permissions for a project*
  - *Integrate WRCS functionality with a new development environment*
  - *Port to a different network-management system*

Scenario Evaluation

Scenario	Direct/Indirect	Required Changes
Modify the user interface in minor ways	Indirect	Modify one or more components that call the win31 API, specifically main, diff, and ctrls.
Change access permissions from a project	Direct	
Integrate with a new development environment	Indirect	Modify hook and add a module along the lines of bcext, mcext, and vbext
Port to a different network-management system	Indirect	Modify wrcs

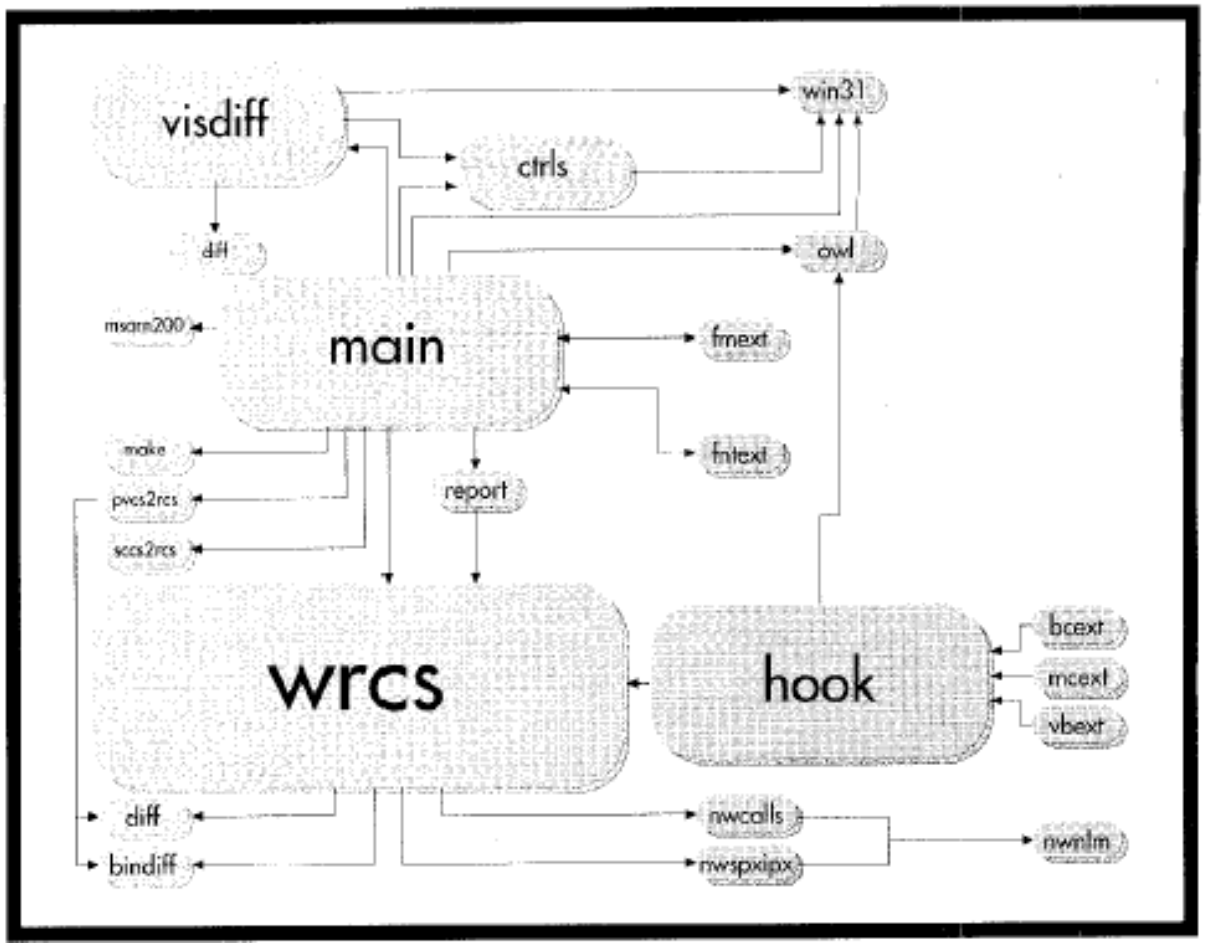
Table 1 B

Interaction By Module

Module	Number of Changes
wrcs	7
main	4
book	4
visdiff	3
ctris	2
report, diff, bindiff, pvcs2rcs, sccs2rcs, nwcalls, nwspxipx, nwnlm	1 each

Table 2

Fish-Eye Representation



Advantages

- Enhanced Communication
- Improvement of Traditional Metrics
- Proper Description Level
- Efficient Scenario Generation
- Deepened Understanding of the System
- Ability to make high-level comparisons of competing designs and to document those comparisons
- Ability to consider and document effects of sets of proposed system changes

References

- SAAM: A Method for Analyzing the Properties of Software Architecture
- Scenario-Based Analysis of Software Architecture

Thank You

How to apply ATAM

(Group 2 and 7)

Billy Alexander (**Example**), Padmaja Havaldar
Fengyou Jia, Cem Oguzhan (**Intro**),
Hulda Adongo , Yang Zheng
Manmohan Uttarwar, Gautham Kalwala (**Conclusion**)

Based upon paper (#29):
"The Architecture Tradeoff Analysis Method (ATAM)", authored by Kazman et al. (1998)

Key features of ATAM

1. Tradeoff Analysis among multiple quality attributes (performance, availability, security, etc.), Looking optimal tradeoff points, rather than optimal individual attributes
2. Iterations of the ATAM (spiral model of design)
3. Trend analysis only (not detailed values)

Steps of the method:

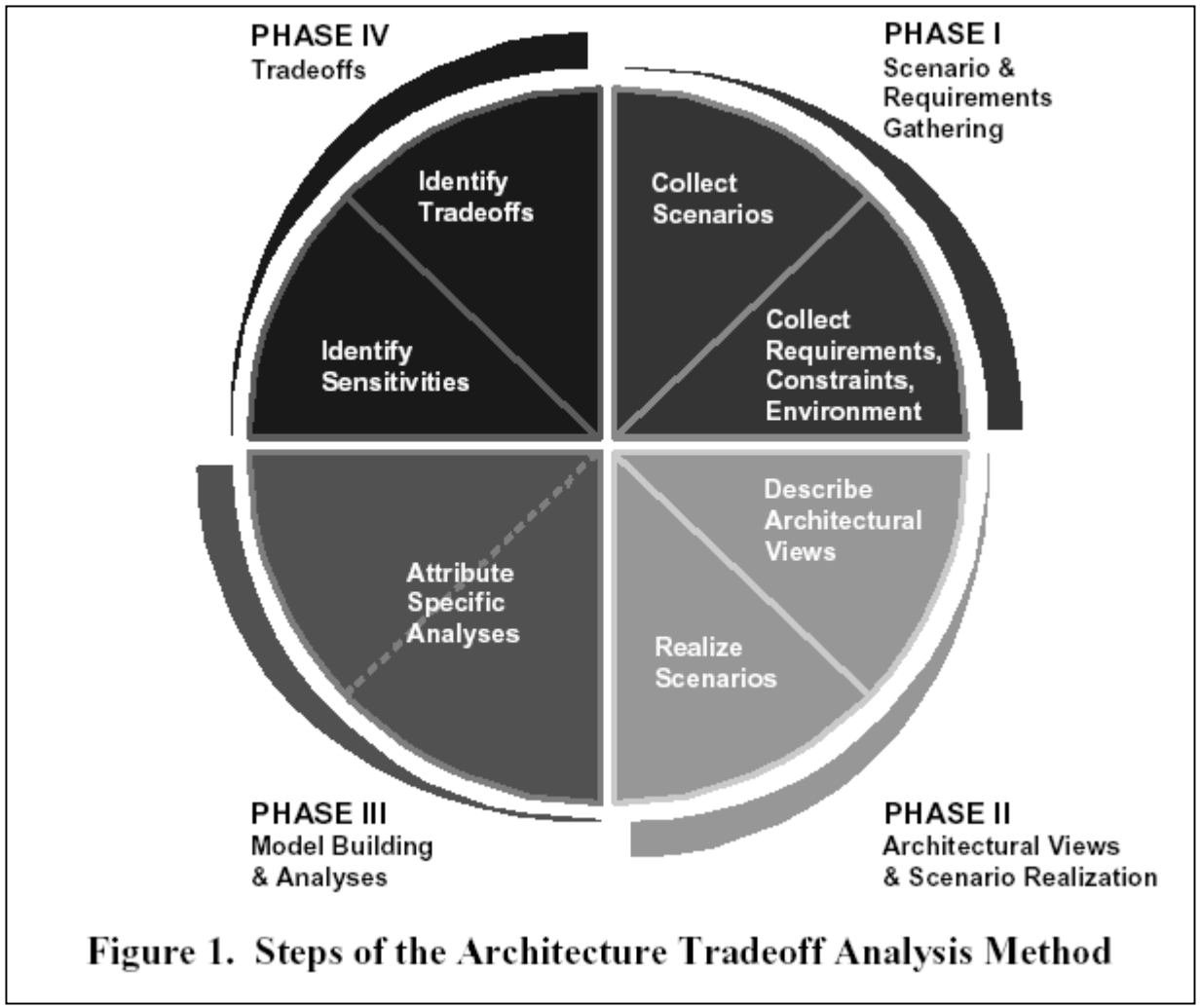


Figure 1. Steps of the Architecture Tradeoff Analysis Method

Iterations of ATAM

1. Following Tradeoff points found (elements that affect multiple attributes)
2. Then we either:
   1) refine the models and reevaluate
   2) or refine the architectures (change the models to reflect these refinements and reevaluate
   3) or change some requirements

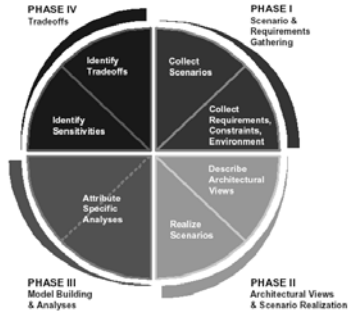


Figure 1. Steps of the Architecture Tradeoff Analysis Method

How to apply ATAM

An Example (ATAM) Analysis:

System Description:

Remote temperature sensor (RTS)

1. measuring the temperature of a set of furnaces through a hardware device (ADC)

2. reporting those temperatures to operators

3. Sending periodic temperature update to hosts

4. Hosts sending control requests to RTS (changing the frequency of updating)

5. Scenarios collection

(1) For performance analysis

- the client sends a control request and receives the first periodic update
- the client receives periodic updates at the specified rate

The availability analysis is also guided by two scenarios,

(2) For availability analysis

- a server suffers a software failure and is rebooted
- a server suffers a power supply failure and the power supply is replaced

6. Collect Requirements/ Constraints/ Environment

(1) Performance requirements

PR1: Client must receive a temperature reading *within F seconds* of sending a control request.

PR2: Given that Client X has requested a periodic update every T(i) seconds, it must receive a temperature *on the average every T(i) seconds.*

PR3: The interval between consecutive periodic updates must be *not more than 2T(i) seconds*.

(2) Availability requirements

AR1: System must not be unavailable for more than 60 minutes per year.

6. Collect Requirements/ Constraints/ Environment

In addition to these requirements, we will assume that the behavior patterns and execution environment are as follows:

- Relatively infrequent control requests
- Requests are not dropped.
- No message priorities
- Server latency = de-queuing time (*Cdq* = 10 ms) + furnace task computation (*Cfnc* = 160 ms)
- Network latency between client and server (*Cnet* = 1200 ms)

7. Describe Architectural Views

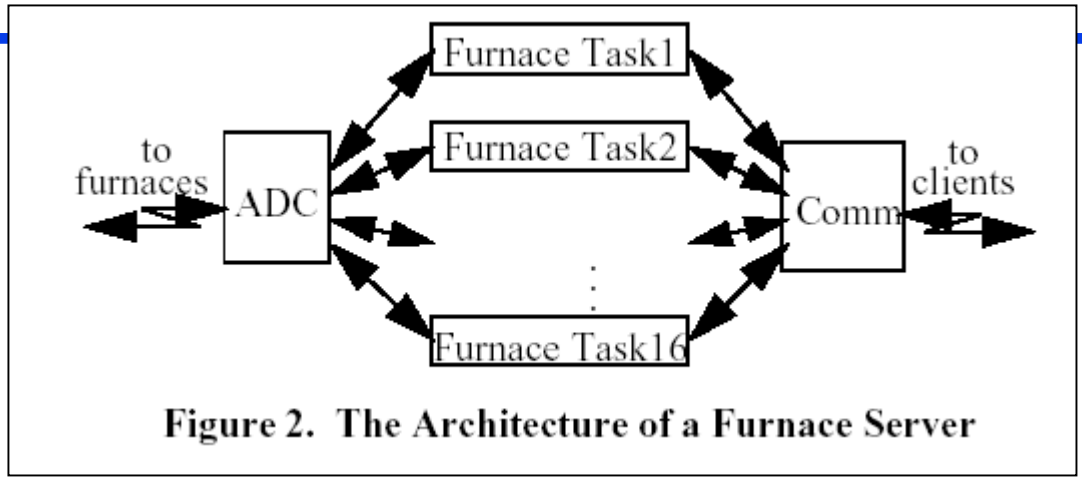


Figure 2. The Architecture of a Furnace Server

7.1 Architectural Option 1 (Client-Server)

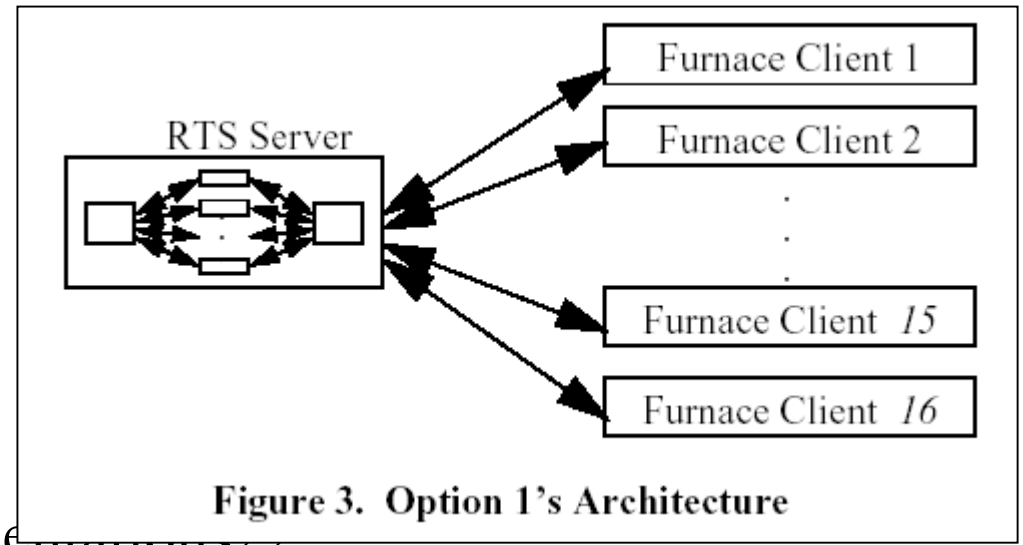


Figure 3. Option 1's Architecture

7.2 Architectural Option 2 (Client-Server-Server)

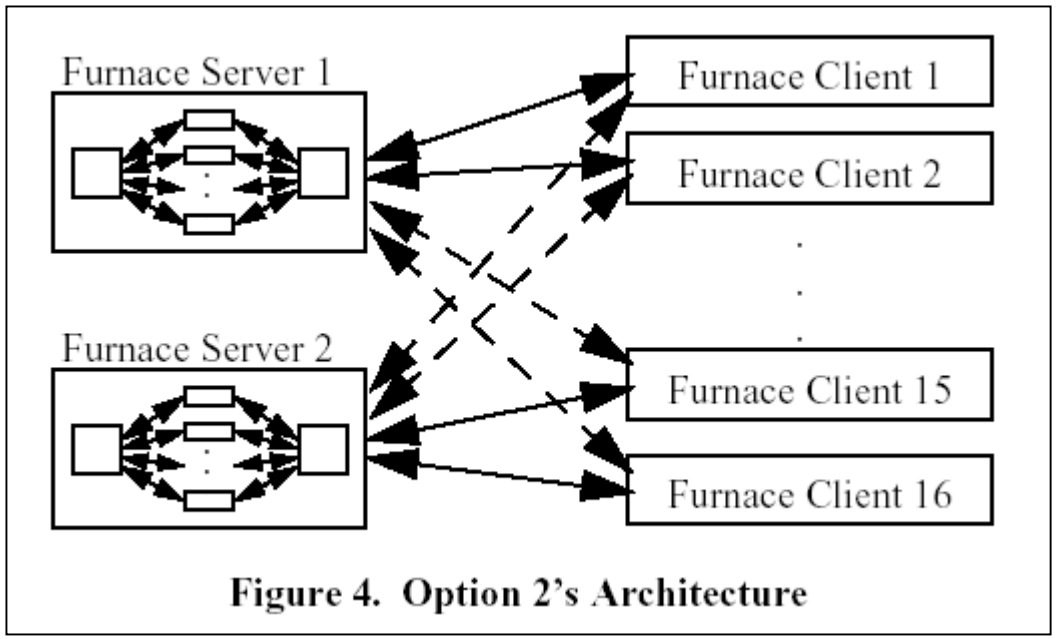


Figure 4. Option 2's Architecture

7.3 Architectural Option 3 (Client-Intelligent Cache-Server)

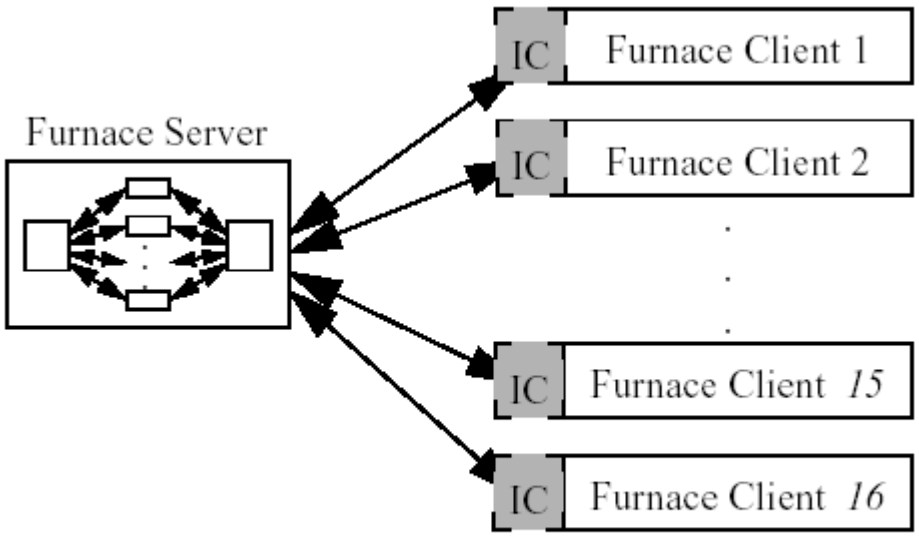


Figure 5. Option 3's Architecture (with Cache)

8. Realize Scenarios/Performance Analyses

8.1 Performance Analysis of Option 1

WCCL	ACPL	Jitter
41,120 ms	5,100 ms	20,400 ms

8.2 Performance Analysis of Option 2

WCCL	ACPL	Jitter
20,560 ms	2,550 ms	9,520 ms

8.3 Performance Analysis of Option 3

WCCL	ACPL	Jitter
41,120 ms	5,200 ms	≤20,400 ms

Notes:

1. WCCL = *worst-case control latency*, ACPL = *average-case periodic latency*, and BCPL = *best-case periodic latency*.

worst case jitter = *WCPL* - *BCPL* = 21,760 - 1,360 = 20,400

Detailed calculations about WCCL, ACPL, and Jitter can be found in reference paper: Babacci et al. 1997.

9. Realize Scenarios/Availability Analyses

9.1 Availability Analysis of Option 1

Repair Time	Failures/yr	Availability	Hrs down/yr
12 hours	24.	0.96817	278.833
10 minutes	24.	0.99954	3.9982

9.2 Availability Analysis of Option 2

Repair Time	Failures/yr	Availability	Hrs down/yr
12 hours	24.	0.99798	17.7327
10 minutes	24.	~1.0	0.0036496

1. Major failures:

e. g., a burned-out power supply, taking 12 hrs to fix.

2. Minor failures:

e. g., SW bugs, taking 10 minutes to repair.

9.3 Availability Analysis of Option 3

Repair Time	Failures/yr	Reliability	Hrs down/yr
12 hours	24.	0.96839	276.91
10 minutes	24.	0.9997	2.66545

Solving the Markov model and getting the results in three tables.

10. Critique on the Options

- Option 1 has **poor performance and availability**. It is also the **least expensive** option (in terms of hardware costs; the detailed cost analyses can be found in [1]).
- Option 2 has **excellent availability**, but at the **cost of extra hardware**. It also has **excellent performance** (when both servers are functioning), and the characteristics of option 1 when a single server is down.
- Option 3 has slightly better availability than option 1, better performance than option 1 (in that the worst-case jitter can be bounded), slightly greater cost than Option 1, and lower cost than Option 2.

11. Sensitivity Analyses

Attributes are sensitive to the number of servers (performance increases linearly as the number of servers increases).

12. Security Analyses

Security requirement:

SR1: The temperature readings must not be corrupted before they arrive to the client.

Security Scenarios:

- A threat object between the server and operators modifies the temperatures received by the operators
- A server is spoofed to misrepresent temperatures received by the operators.

So, to calculate the probability of a successful attack within an acceptable window of opportunity for an intruder, we define initial values that are *reasonable* for the functions provided in the RTS architectures. These values are shown in Table 7:

Attack Components	Value
Attack Exposure Window	60 minutes
Attack Rate	0.05 systems/min
Server failure rate	24 failures/year
Prob of server failure within 1 hour	0.0027

Attack Components		Value
Prob of successful	TCP Intercept	0.5
	Spoof IP address	0.9
	Kill Connection	0.75
	Kill Server	0.25

Table 7. Environmental Security Assumptions

Three possible ways to succeed for spoof-server attack:

Attack Type	Expected Intrusions in 60 Mins
Kill Connection	2.04
Kill Server	0.66
Server Failure	0.0072

Table 8. Anticipated Spoof-Attack Success Rates

Critiques:

Table 8 showed that the possible intrusion rates were much higher than expected (requirements), Thus, to refine architecture options is required (need another iteration).

12.1 Refined Architectural options

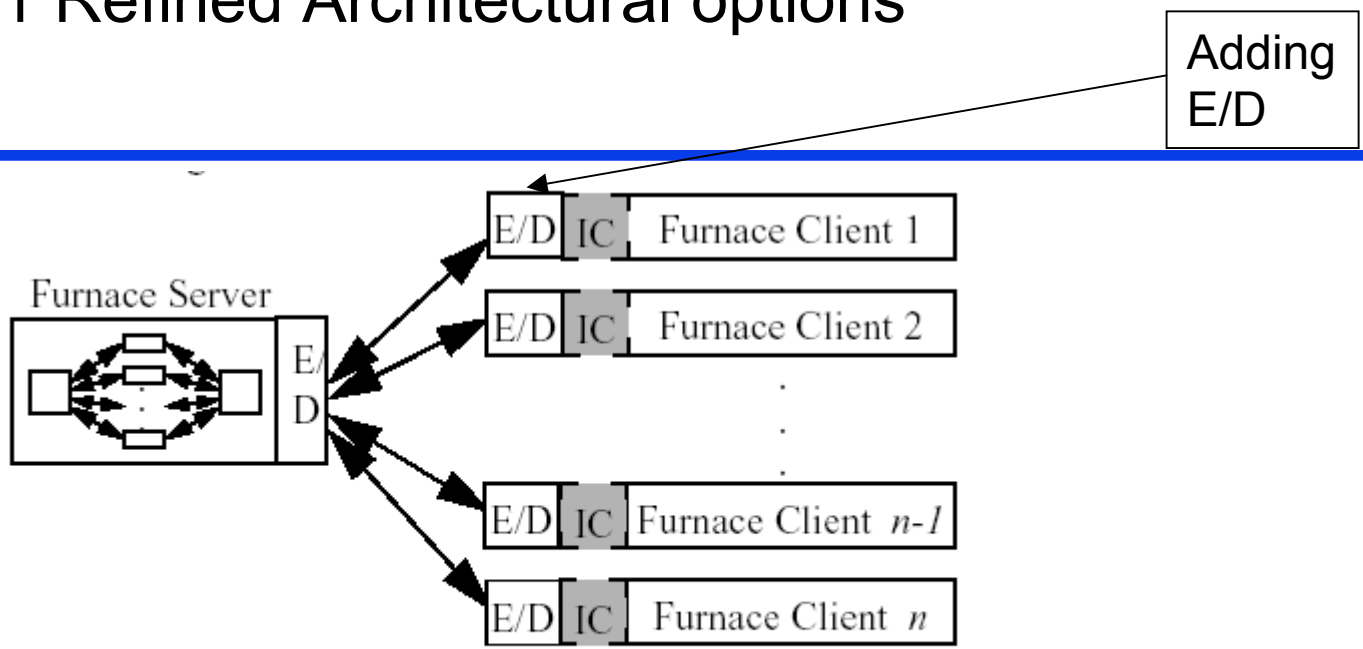


Figure 7. Security Modifications

Note: E/D (Encryption/Decryption)

(A most common security "bolt-on" solution, unreasonable change range generated by an intruder will be deemed (recognized) due to the addition of E/D)

Attack Components		Value
Prob of successfu	Decrypt	0.0005
	Replay	0.05
	Key Distribution	0.09

Table 9. Additional Security Assumptions

Attack Type	Expected Intrusions in 60 Mins
Kill Connection	0.18225
Kill Server	0.03375
Server Failure	0.0006

Table 10. Spoof Success Rates with Encryption

Attack Type	Expected Intrusions in 60 Mins
Kill Connection	0.16875
Kill Server	0.05625
Server Failure	0.005

Table 11. Spoof Success Rates with Intrusion Detection

Attack Type	Expected Intrusions in 60 Mins
Kill Connection	2.04
Kill Server	0.66
Server Failure	0.0072

Table 8. Anticipated Spoof-Attack Success Rates

(Before adding E/D)

13. Sensitivities and Tradeoffs

At this point, we have discovered an *architectural* tradeoff point in the number of servers. Performance and availability are correlated positively, while security and presumably cost are correlated negatively with the number of servers. We cannot maximize cost, performance, availability, and security simultaneously.

Conclusions

RTS Case Study

- Vague requirements & architectural options.
- Useful characteristics.
- Costs & benefits.
- Trade-offs.
- Develop informed action plans.
- Evaluations & iterations.

CONCLUSION : ATAM

- Motivated by rational choices among the competing architectures.
- Concentrates on identification of trade off points.
- Early clarification of requirements.
- Enhanced understanding and confidence in systems ability to meet the requirements.
- This method (ATAM) is still under development in SW engineering.

Key references

1. Babacci et al. (1997). CMU/SEI-97-TR-29. Pittsburgh, PA: Software Engineering Institute.

1. Dobrica et al. (2002). IEEE Transactions on Software Engineering, Vol. 28 NO. 7, July.

2. Kazman et al. (1999). Proc. Int'l Conf. Software Eng. (ICSE '99), pp. 54-63, May.

3. Kazman et al. (1998). Proc. Fourth Int'l Conf. Eng. Of Complex Computer Systems (ICECCS '98), Aug.

S/W Architecture Re-engineering

Zhigang Xie Ryan Young Ravi Athipatla

Jinhua Wang Shufeng Li

Vishal Solipuram Feng Chen Krishan Narasimhan

What is SBAR?

- Abbreviation for Scenario-based Architecture Re-engineering

- "SBAR estimates the potential of the designed architecture to reach the software quality requirements."
  - L. Dobrica: *A Survey on Software Architecture Analysis Methods*

Importance of SBAR

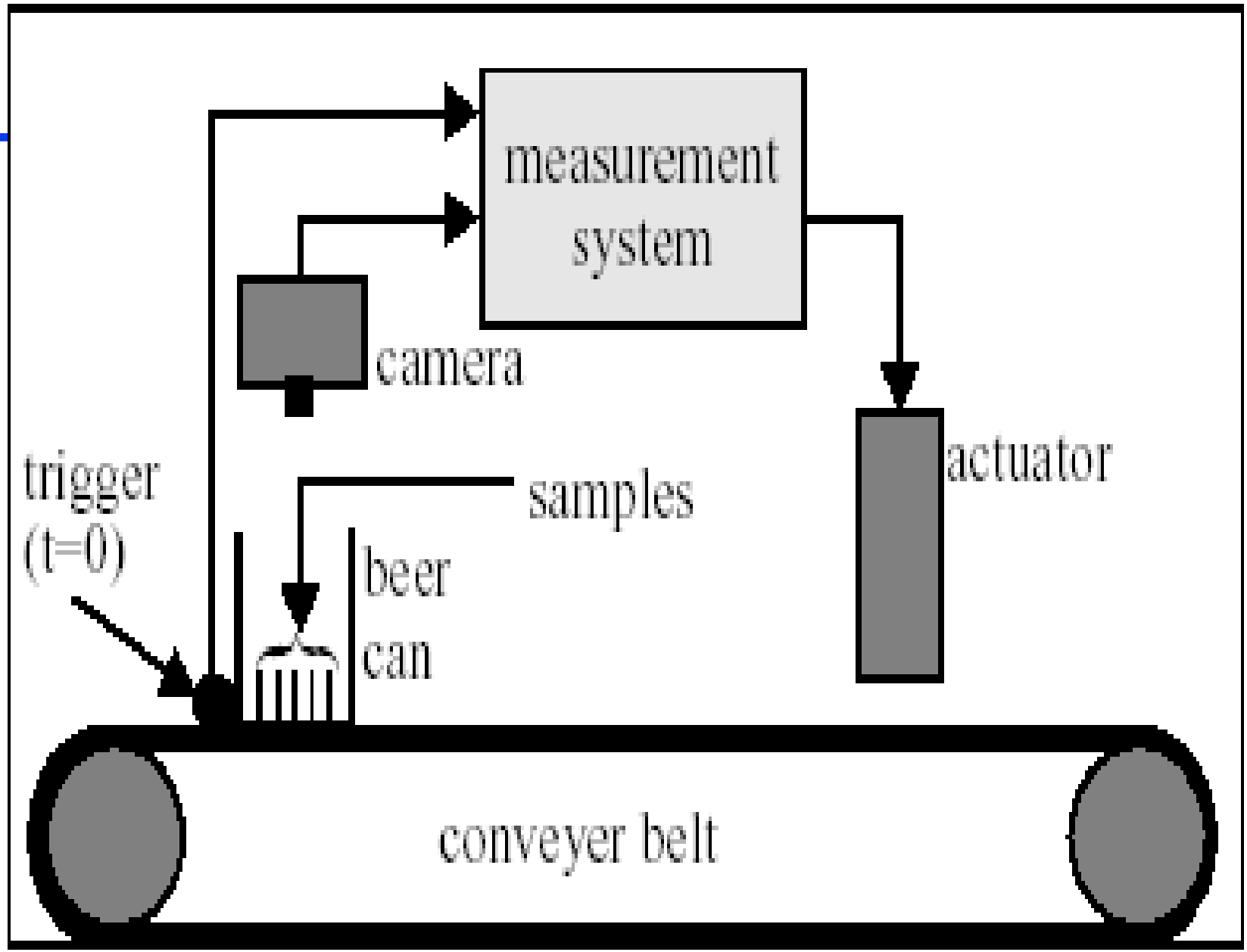
- A system is never a pure real-time system, or a fault-tolerant system, or a re-usable system.
- Single non-functional requirement (NFR) is not a satisfactory measurement, since NFRs often conflict.
- In a realistic system a balance of NFRs is needed for an accurate assessment of a software architecture.

Assessing Quality Attributes

1. Scenarios
2. Simulation
3. Mathematical Modeling
4. Experience-based Reasoning

An Example.....

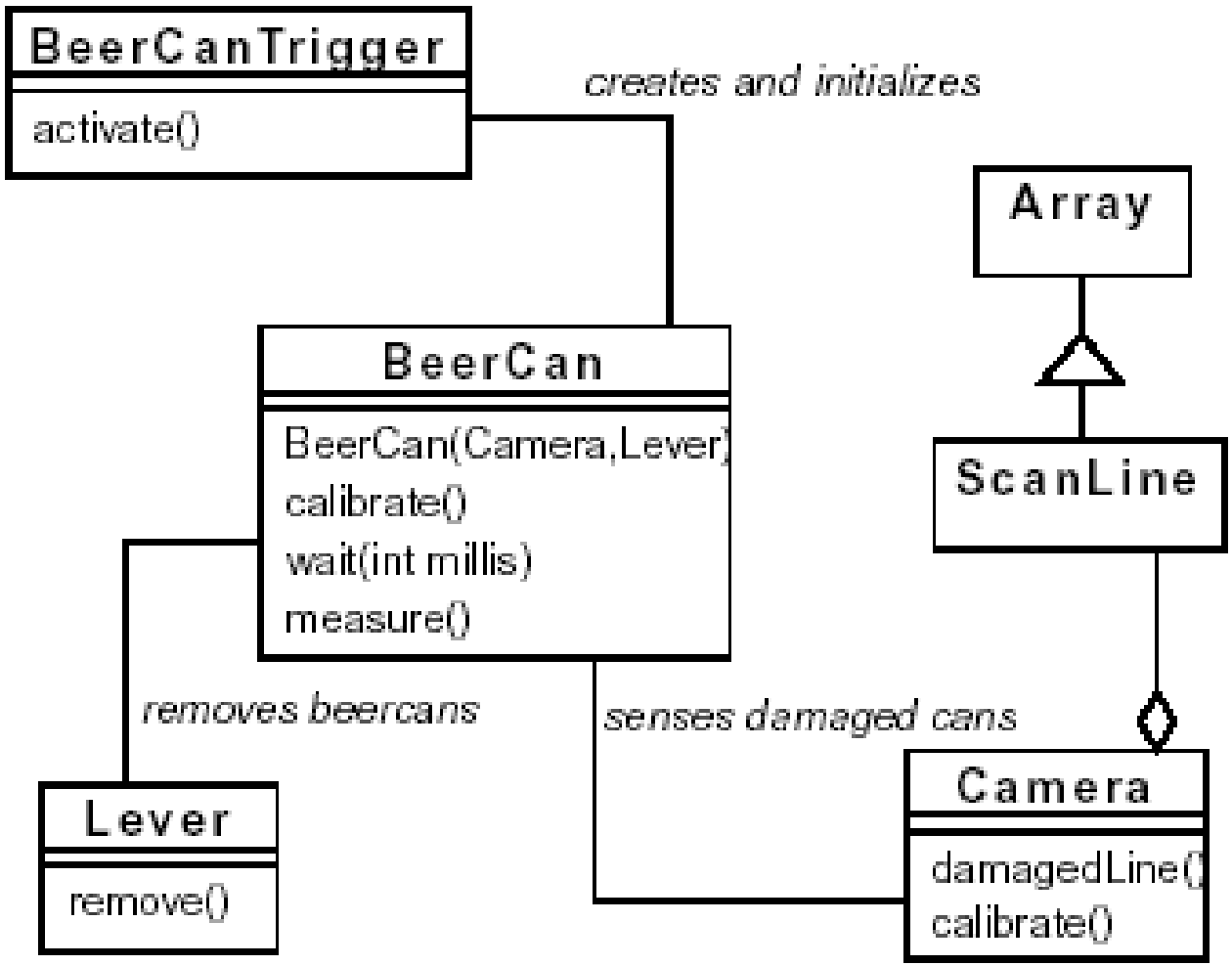
- Beer Can Inspection System:
  - To illustrate the architecture reengineering method, a beer can inspection system is used.
  - The inspection system is placed at the beginning of beer can filling process and its goal is to remove dirty beer cans from the input stream. Clean cans should pass the system without any further action.
  - The system consists of a triggering sensor, a camera and an actuator that can remove cans from conveyer belt.


measurement
system
camera
actuator
trigger
(t=0)
samples
beer
can
conveyer belt

Functions

- When a can is detected, the system receives a trigger from a hardware trigger.
- After a predefined amount of time, the camera samples an image of the can. This sampling is repeated a few times and subsequently the measured images are compared to ideal images and a decision about removing or not removing the can is made.
- If the can should be removed, actuator is invoked at a point in time relative to the point in time when the trigger event took place.

Object Model

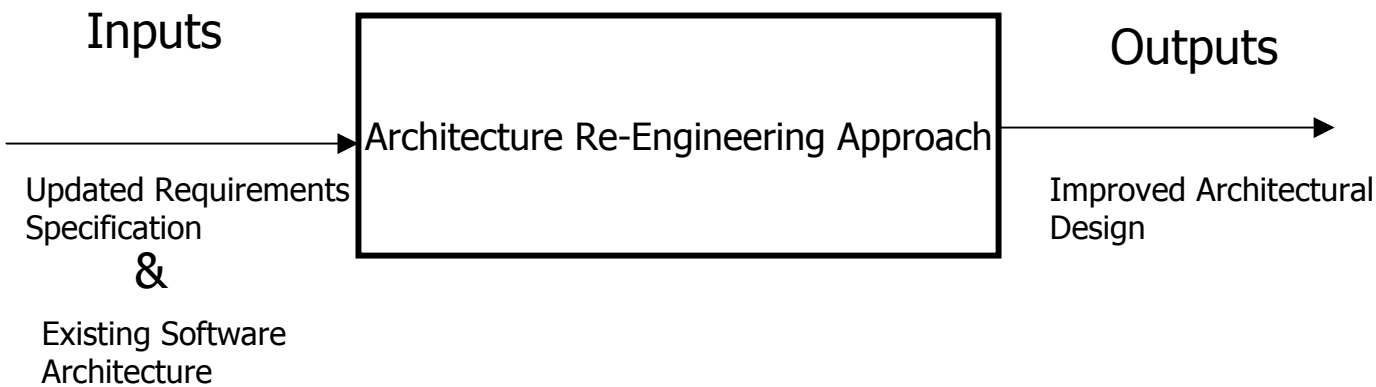


Author's Experience

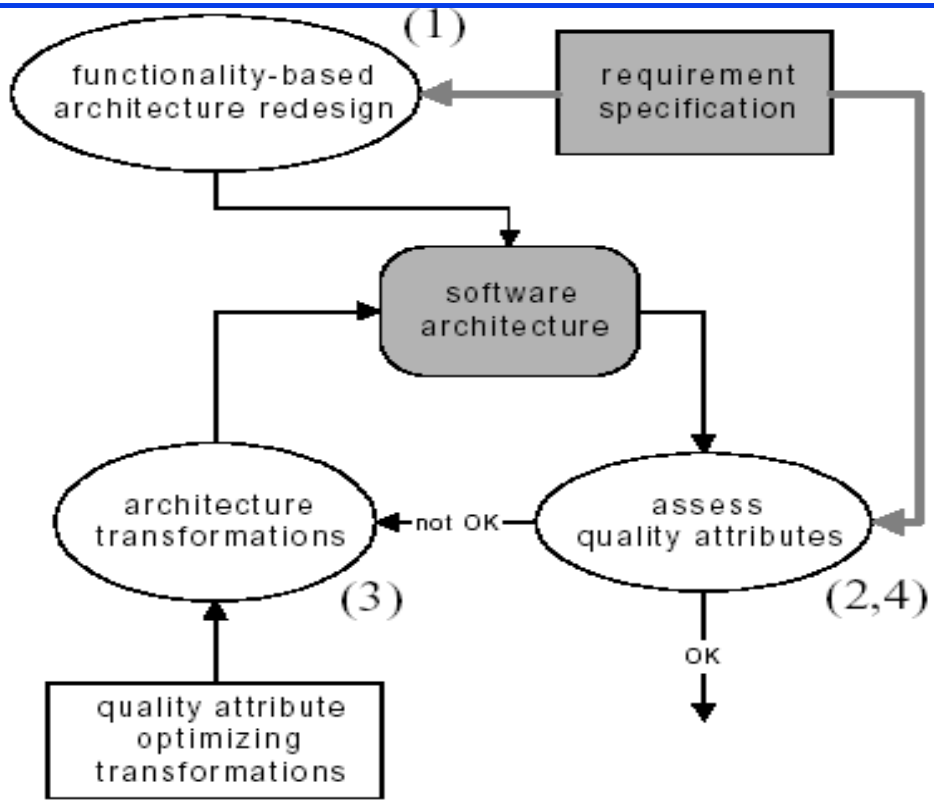
- Generally we handle S/W quality requirements by an informal process.
- If found short-comings, then re-design iteratively over system development, but this proves very costly.
- S/W quality requirements often conflict
  - Real-time Vs Reusability
  - Flexibility Vs Efficiency
  - Reliability Vs Flexibility
- Conventional design methods focus on a single quality attribute and treat all others as having secondary importance.

Architecture Re-engineering Method

- S/W engineers need to balance the various quality attributes for any realistic system.
- The authors propose an architectural re-engineering method that provides an objective approach.



Method Outlined……



1. Incorporate new functional requirements in the architecture
2. Software quality assessment
3. Architecture transformation
4. Software quality assessment

Assessment

- Assessing Software Quality Requirements

1. Scenario-based evaluation: Develop a set of scenarios that concretize the actual meaning of the attribute. Useful for development related S/W qualities like reusability and maintainability

2. Simulation: Complements scenario-based evaluation. Is useful for evaluating operational software qualities like performance or fault-tolerance.

3. Mathematical Modeling: Allows for static evaluation of architectural design models.

4. Experience-based reasoning: Evaluation process is less explicit and more based on subjective factors as intuition and experience.

Transformation

Iterative Steps :-

- Complete architecture design.
- Compare with the requirements.
- Then update architecture.

Note :-

- The transformations made are minor.
- The functionality does not change, only the quality attributes change.
- It is not feasible to start bottom-up during design and reengineering.

Different Approaches

- Impose architectural style. e.g.. layered architectural style
- Impose architectural pattern.
- Apply design pattern.
- Convert quality requirements to functionality.
- Distribute requirements.

S/W Quality Requirements

- Functional requirements generally can be evaluated relatively easy by tracing the requirements in the design.
- On the other hand, S/W quality requirements are much harder to assess.
- Few such quality requirements are:
  - Reusability
  - Maintainability
  - Real-time
  - Robustness
- As mentioned previously, development related S/W qualities are easiest assessed using scenarios.

Reusability

- This quality attribute should provide a balance between generality and specifics.
- The architecture and its components should be general since they should be applied in other similar situations.
- The architecture should provide concrete functionality that provides considerable benefit when it is reused.
- Five scenarios that are tested in this article:
  - R1: Product packaging quality control
  - R2: Surface finish quality control
  - R3: Quality testing of micro-processors
  - R4: Product sorting and labeling
  - R5: Intelligent quality assurance system

Maintainability

- The goal here is that the most likely changes in requirements are incorporated in the software system against minimal effort.
- Five scenarios that are tested in this article are:
  - M1: The types of input or output devices used in the system is excluded from the suppliers assortment and need to be changed, by the S/W.
  - M2: The S/W needs to be modified to implement new calculation algorithms.
  - M3: The method of calibration is modified.
  - M4: The external systems interface for data exchange change.
  - M5: The hardware platform is updated, with new processor and I/O interface.

Applying SBAR

Iterative process until quality requirements are met:

- Evaluate software quality attributes of the application architecture
- Identify the most prominent deficiency
- Transform the architecture to remove the deficiency

Evaluation
How much re-use is possible?

- How much will I be able to reuse the software
- Ratio of Re-used components '*as-is*' to the total number of components
- As close to 1 as possible
- Presence of high coupling limits the possibility of re-use

Evaluation
Effort needed to maintain

- How easy is it to fix
- Ratio of Affected components to Total components
- As close to 0 as possible
- Changes usually require many components to be modified

Transformations
Component-level

Problem:

New item type requires the source code of most components to be changed

Transformation:

specific type → generic type

Result:

Improves reusability and maintainability

Transformations
Abstraction

Problem:

Type dependence at component creation

Transformation:

Use Abstract Factory pattern

Results:

Improves maintainability

Transformations
Choose Strategy

Problem:

Changes have to be made in every component performing similar task

Transformation:

Apply the Strategy pattern

Results:

Gained maintainability outweighs loss in reusability

Decrease Dependence on Global State

Problem:

Calibration of the measurement system

Transformation:

Introduce calibration strategy

Results:

Improves maintainability

Reduce Coupling between calibration & measurement

Problem:

Coupling between calibration strategy and the measurement item.

Transformation:

Apply Prototype design pattern.

Results:

Improves maintainability and reusability.

Software Quality		Iteration no.					
	Scenario	0	1	2	3	4	5
Reusability	R1	2/4	3/5	4/6	3/9	3/9	4/10
	R2	2/4	3/5	4/6	3/9	3/9	4/10
	R3	0/4	0/5	1/6	2/9	2/9	3/10
	R4	0/4	0/5	1/6	2/9	2/9	3/10
	R5	0/4	0/5	1/6	2/9	2/9	3/10
Maintainability	M1	1/4	1/5	1/6	2/9	3/9	2/10
	M2	4/4	4/5	3/6	2/9	3/9	2/10
	M3	4/4	5/5	6/6	9/9	2/9	2/10
	M4	4/4	3/5	3/6	3/9	3/9	3/10
	M5	4/4	5/5	6/6	9/9	9/9	10/10

Table 1: Analysis of architecture

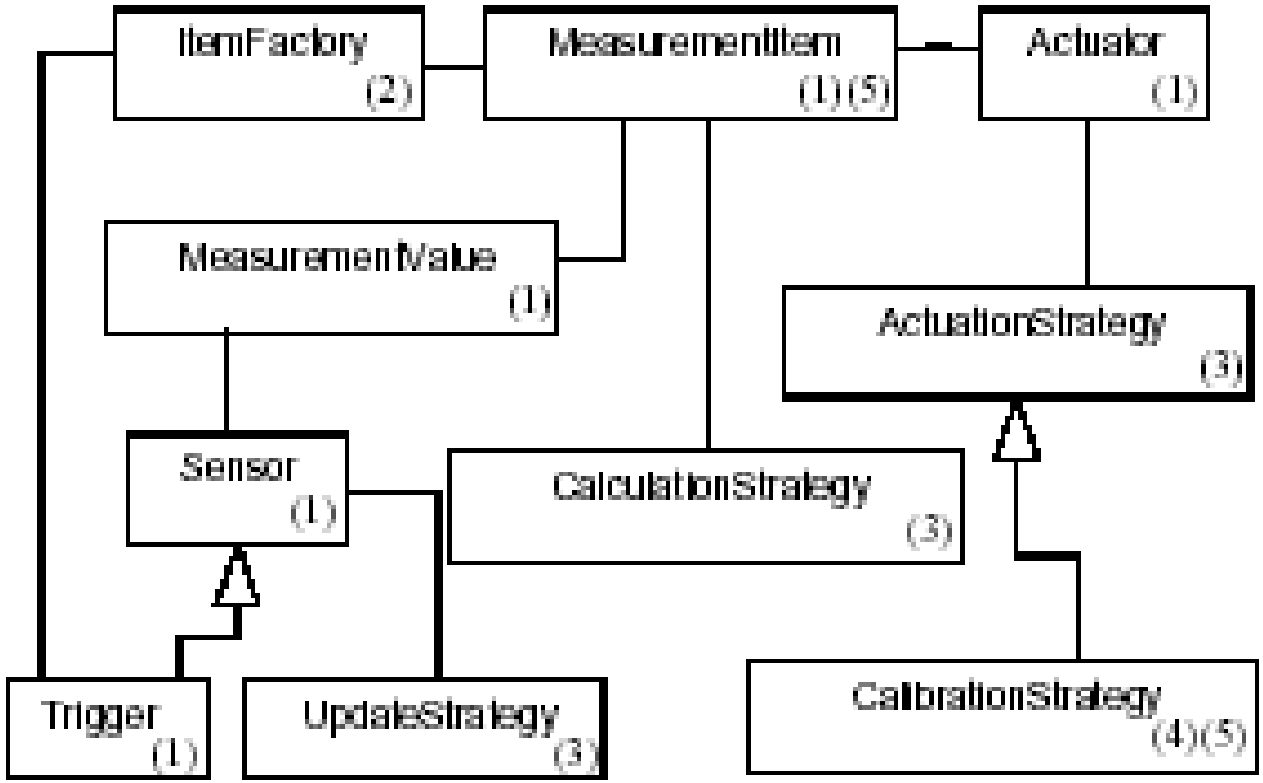


Figure 5. The Measurement Systems DSSA[1]

Evaluation

- Overall, the result from the transformations is satisfying and the analysis of the scenarios shows substantial improvement (author's conclusion).
- Each iteration seems to solve a problem concerning some attribute. The drawback may be that, we do not have a prior idea of how many iterations it is going to take.
- Identifying all possible problems that may lead to difficulties in re-use and maintainability is a challenging task in itself.

References

- *A Survey on Software Architecture Analysis Methods*, L. Dobrica

- *Scenario-based Software Architecture Re-engineering*, PerOlof Bengtsson & Jan Bosch