

# Software Architecture Design with ArchE

Software Engineering Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213

Author Felix Bachmann, Len Bass,  
Phil Bianco  
Date 03/26/07



Software Engineering Institute

Carnegie Mellon

© 2007 Carnegie Mellon University

# Software Engineering Institute

Federally funded Research and  
Development Center

Created in 1984

Under contract to  
Carnegie Mellon University

**Mission:** Improve the practice  
of software engineering



Software Engineering Institute

Carnegie Mellon

The Architect and ArchE  
Bachmann, Bass, Bianco 03/26/07

© 2007 Carnegie Mellon University

# Introduction

---

The goal of our work in software architectures is to understand the mechanics behind creating good architectures and make this knowledge public.

In collaboration with the Bosch Research and Technology Center in Pittsburgh (Bosch-RTC) we addressed the question:

*Is it possible to codify architectural knowledge in a tool that provides the right information at the right time to the architect?*

The answer is ...



# Contents

---

## The Architecture Design Problem

- Managing the Universe - Dealing with an infinite number of alternatives

## The Architecture Design Solution

- Managing the Imagination - Defining The Requirements
- Managing the Reality - Creating Quality Attribute Models
- Managing the Alternatives - Closing in on the Solution

## Improving the Architecture Design Solution

- Managing the Future - Increasing available and codified knowledge





# Managing the Universe:

*Dealing with an infinite? number of alternatives*



# The Architecture Design Process

---

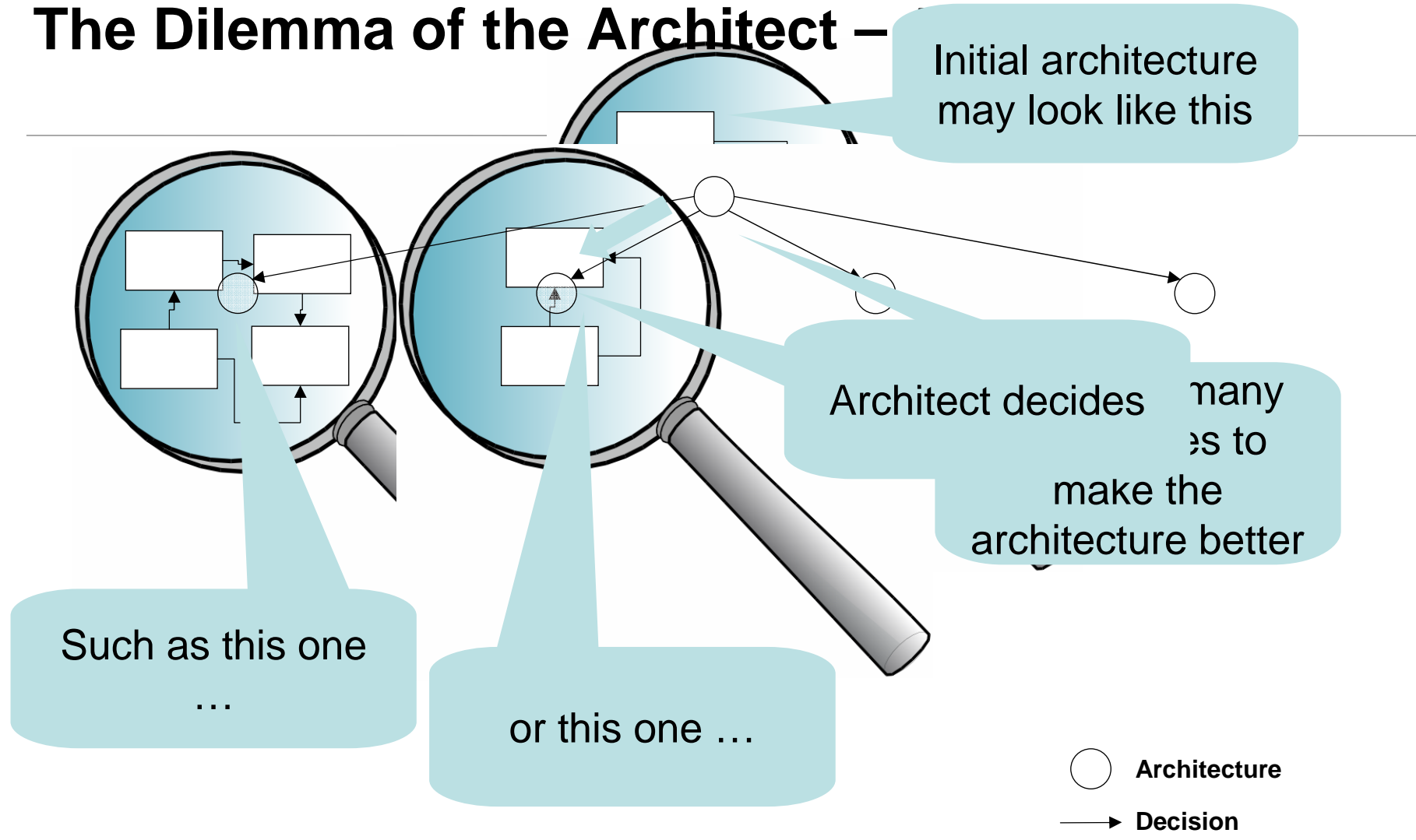
An architecture design follows (should, really!) this process:

1. Create a measurable specification of quality attribute requirements that need to be supported by the architecture
2. Evaluate if the current architecture you have fulfills those requirements
3. If not, make some changes to the architecture to improve and repeat step 2
4. If yes, Lucky you! You are done.

As simple as this may sound, it creates a huge problem ...



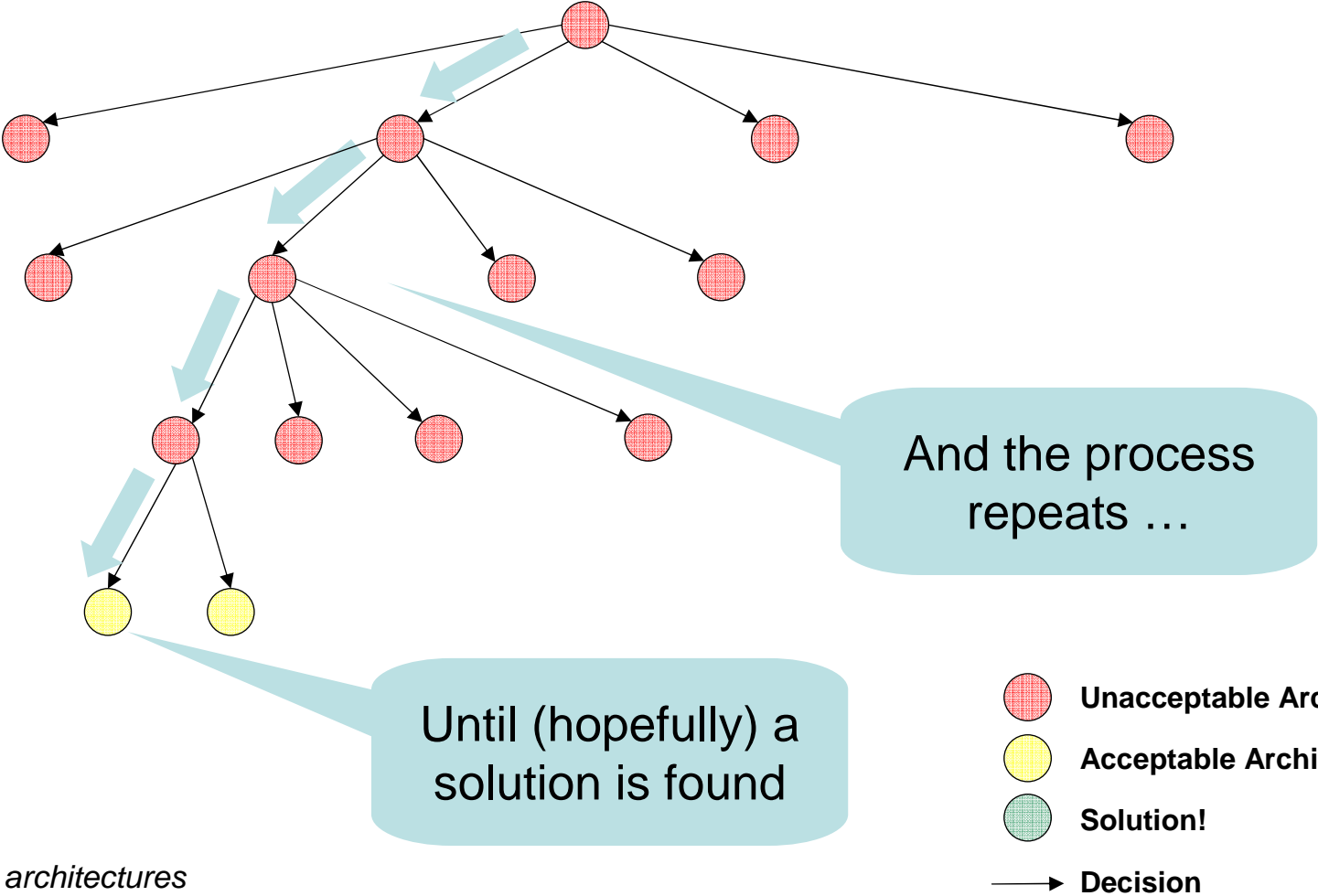
# The Dilemma of the Architect –



*A view of possible architectures*



# The Dilemma of the Architect – 2

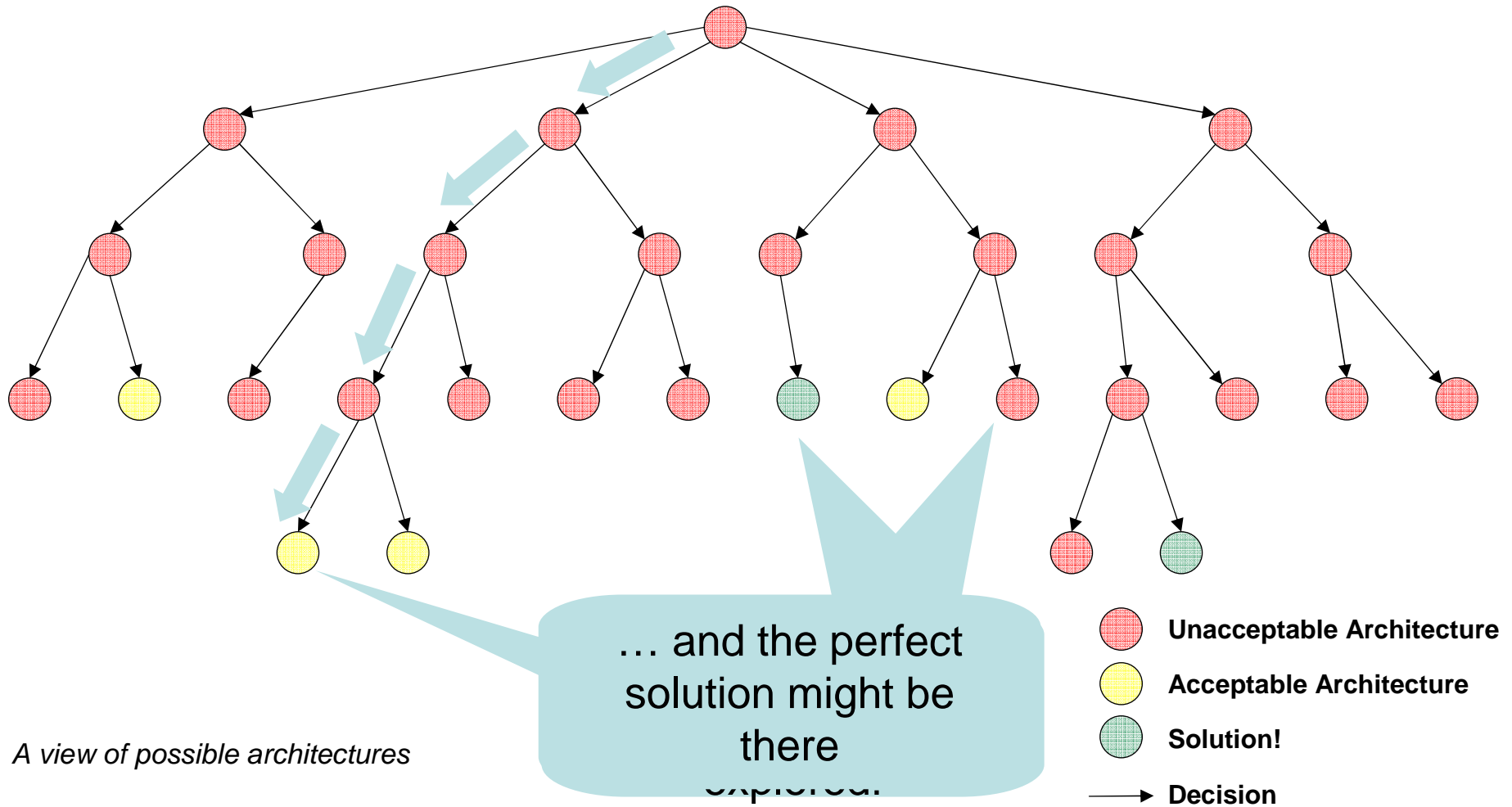


A view of possible architectures





# The Dilemma of the Architect – 3

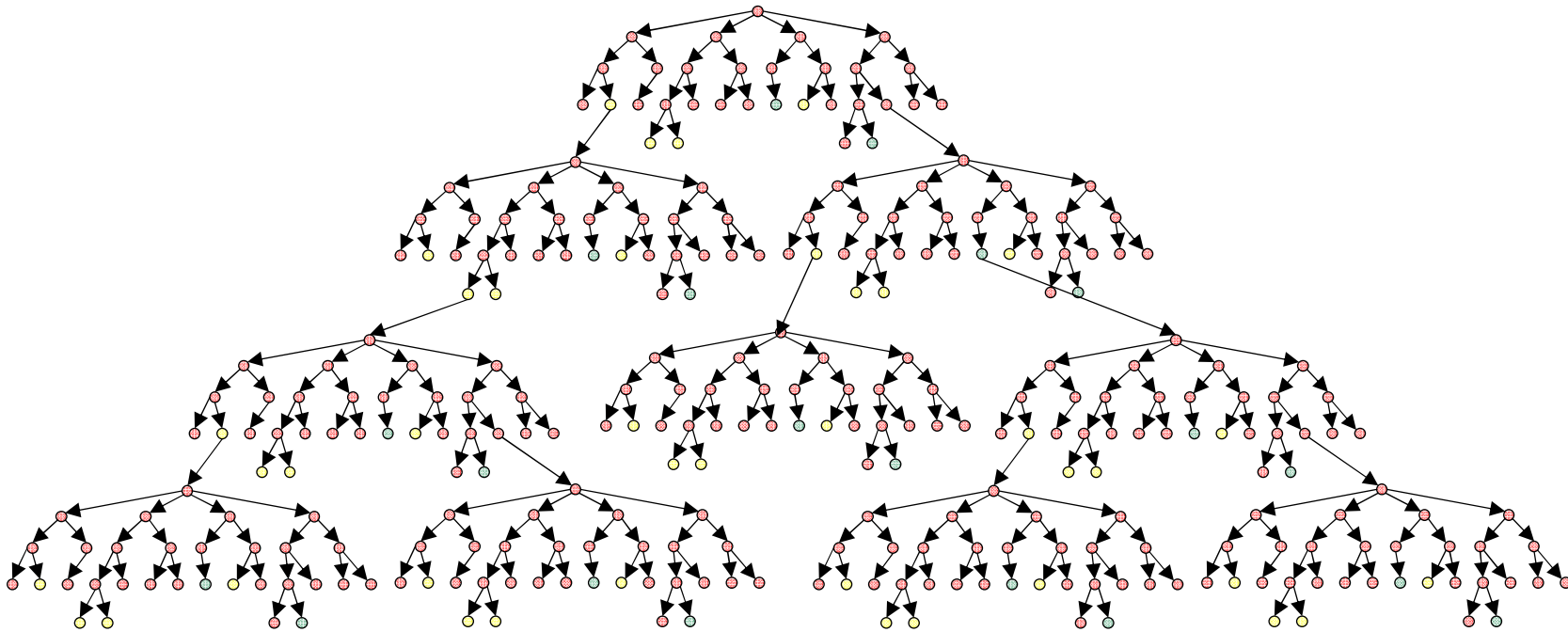


*A view of possible architectures*



# The Dilemma of the Architect – 4

---



*The space of possible architectures for even a simple example is huge*

*It is like finding the needle in a haystack*



# Approach to the Problem

---

Designing an architecture efficiently means to quickly navigate through the vast space of possible architectures.

Some strategies can be used to make it more feasible to actually find a good solution:

- Reduce the number of possible solutions
- Avoid dead ends, paths that do not lead to a solution
- Use tools to point you in the right direction



# Reducing the Possibilities

---

*Software architectures mainly depend on quality attribute requirements*

The quality attributes an architecture has to support determine the architectural elements with their properties, the connections between them with their properties.

The functional requirements determine the responsibilities of the architecture elements and their concrete interfaces

*It is sufficient to focus on quality attributes when searching for a good software architecture*

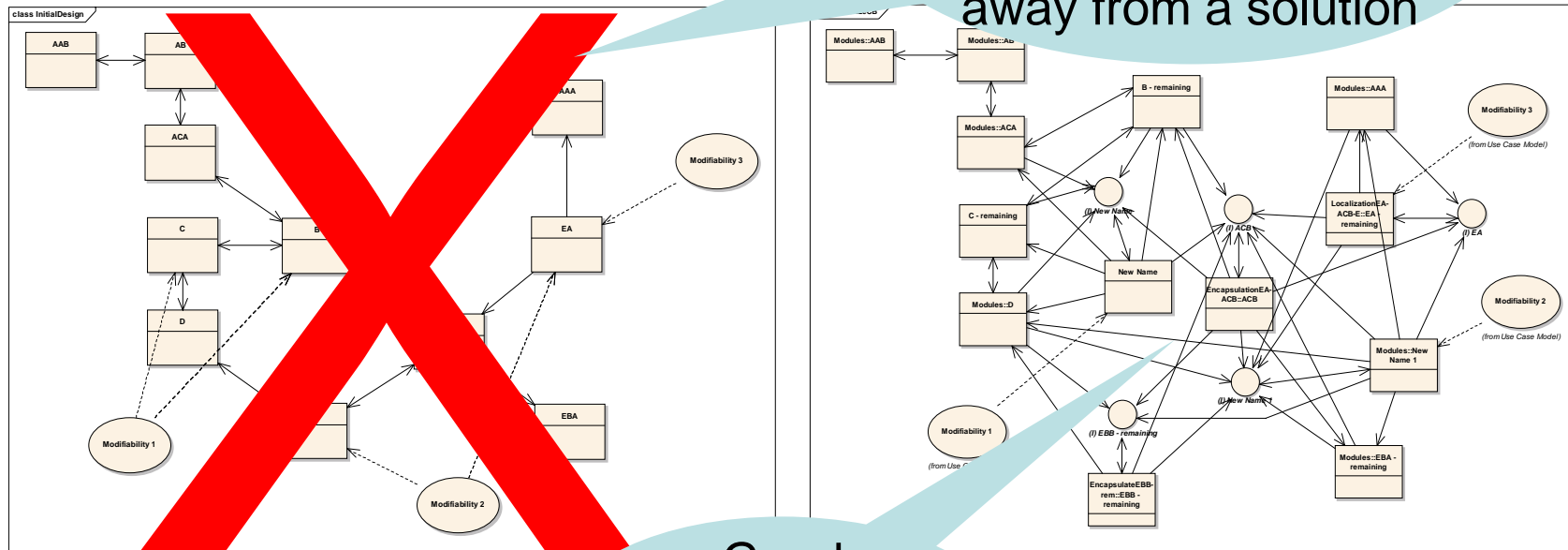
This greatly reduces the space in which to look for an appropriate software architecture.



# Don't Be Fooled By "Good Looking" Solutions

Which design is the better one?

Intuition is not always correct  
May lead architect away from a solution



Design A

Good Solutions are complex

Design B

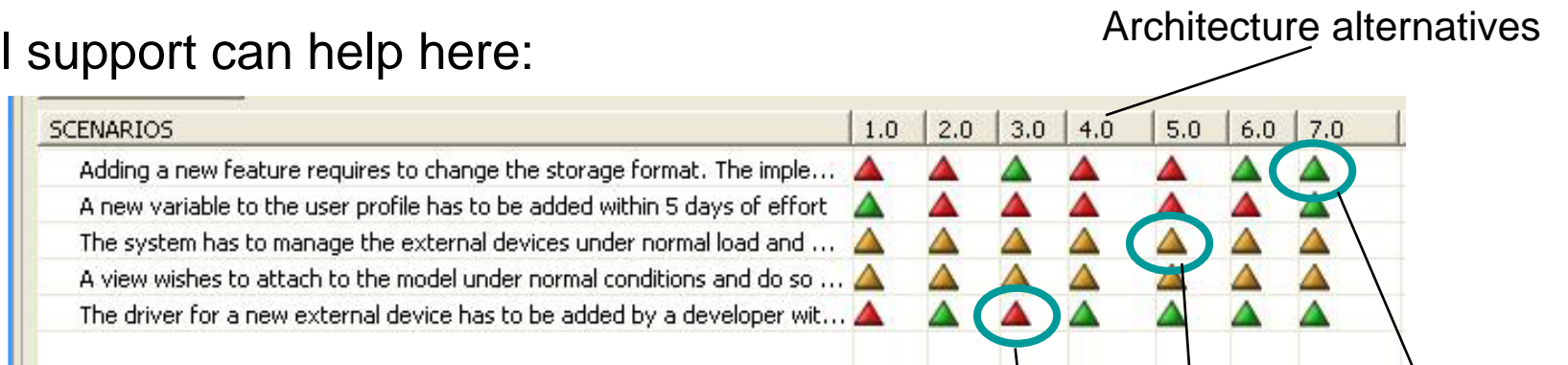


# Pointing in the Right Direction

More architecture alternatives can be explored if

- Evaluation of possible architectures is faster
- recognition and elimination of conflicts between quality attributes requirements are done as soon as they appear – to not end up in a dead end

Tool support can help here:



ArchE explored 7 (simple!) architectures in 2 sec

Deterioration

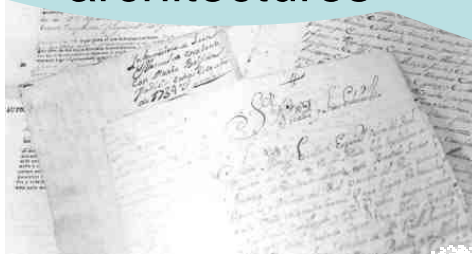
Neutral

Improvement

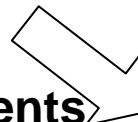


# Who is ArchE?

Your friendly guide helping you navigate through the space of architectures



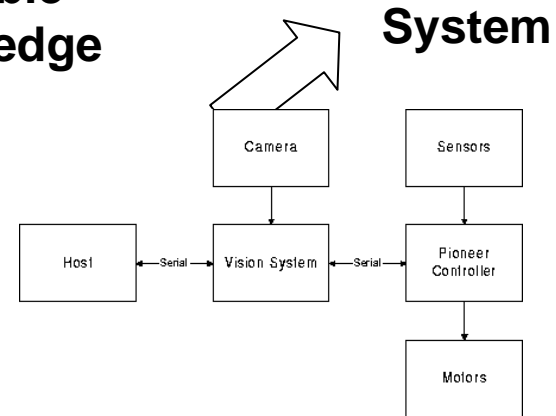
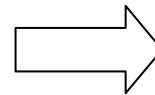
Requirements  
in various  
forms



Available  
knowledge



Designer



Architecture



# What does ArchE do?

---

ArchE is a tool designed to provide useful information about a current architecture to the architect to find a good solution for a given problem.

Such a tool needs to:

- Understand the design process
- Understand quality attributes and their connection to software architectures

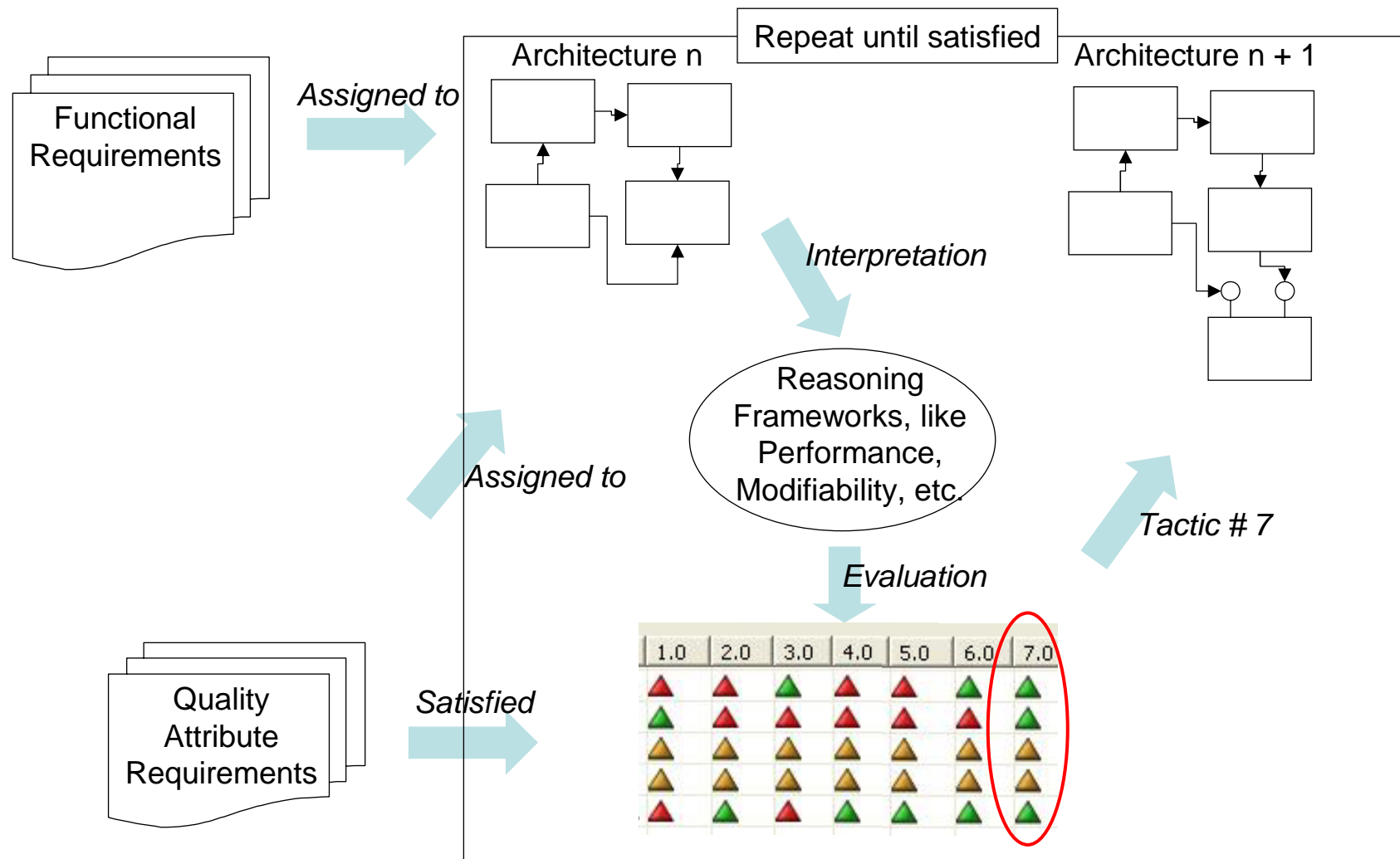
The architect provides the domain knowledge.

Don't be fooled by this nice presentation! ArchE is still a prototype, sorry.





# The Principles of Architecture Design



# The Principles of Architecture Design – 2

---

What to do? – Get your requirements ready:

- The functional requirements
- The dependencies between them
- The quality attribute requirements
- The initial design (can be a design containing just one element, the system)

How do I know my design is good? – Produce quality attribute models that provide you with information about quality attributes.

- Extract required model information from the design (Interpretation)
- Run the model to calculate the values for the quality attribute requirements (Evaluation)

How to improve? – Have a set of tactics that improve the architecture

- Interpret the model to determine plausible tactics
- Apply the tactics to the design by changing elements, relations and their properties





# Managing the Imagination:

*Defining The Requirements*



# Requirements

---

To design a system we need:

- The functional requirements
- The dependencies between them
- The quality attribute requirements

Let us talk a little bit about quality attribute requirements ...



# Quality Attribute Scenarios

---

Quality attributes have to be specified **precisely** using quality attribute scenarios.

A fully specified quality attribute scenario consists of six parts:

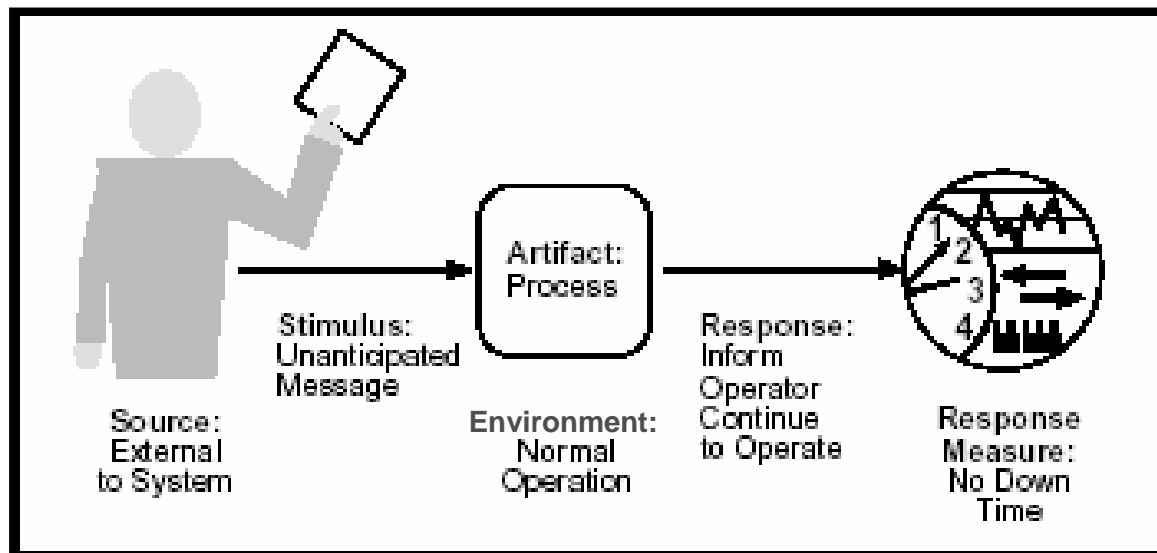
- **Stimulus** – event that is effecting the system
- **Response** – activity as a result of the stimulus
- **Source of stimulus** – The entity that generated the stimulus
- **Environment** – the condition under which the stimulus occurred
- **Artifact stimulated** – the artifact that was stimulated
- **Response measure** – the measure by which the system's response will be evaluated



# Quality Attribute Scenarios

## Example Availability Scenario:

*An unanticipated external message is received by a process during normal operation. The process informs the operator of the receipt of the message and the system continues to operate with no down time.*



For more information:  
Software Architecture in  
Practice, 2<sup>nd</sup> Edition



# Example

---

Throughout this tutorial we use the following example to illustrate the architecture design.

## **The Clemson Transit Assistant System (CTAS)**

- Itinerary planning system
- Plan routes and modes of transportation
- PDA based



# Example – Performance Scenarios

---

An external devices (e.g. GPS) is connected and the system has to operate the device under normal load in under 5 seconds response time.

A user selects a view and this view becomes available and displays the correct data in under 1 second.

The user modifies his/her profile under normal conditions and the profile is modified in under 5 seconds.

The user requests an itinerary under normal conditions and the itinerary is shown in under 5 seconds.

The user saves the current data on the screen under normal conditions and the data is saved in under 10 seconds.





# Example – Modifiability scenarios

---

A new feature requiring a change to the storage format is added. The implementation of the new format has to be done within 3.5 days

A new variable to the user profile is added by an experienced developer within 5 days of effort

The driver for a new external device (e.g. GPS) has to be added by a developer within 10 days



# Functional Requirements

---

Functional requirements become responsibilities that are assigned to elements in the architecture design.

As a starting point all functional requirements are translated one-by-one into responsibilities.

During the design process responsibilities may be refined or split into several responsibilities.

Functional requirements also have dependencies between them, which translates into dependencies between responsibilities.



# Example – Responsibilities with Dependencies

Responsibility uses Responsibility is used by		Attach to Model	Create User Profile	Handle User Interactions	Locate Services	Manage External Devices	Manage Itineraries	Modify User Profile	Query Data	Register Views	Save Data	Show Itineraries
		1	2	3	4	5	6	7	8	9	10	11
+ Attach to Model	1	.								1		
+ Create User Profile	2		.	1								
+ Handle User Interactions	3			.								
+ Locate Services	4				.			1				
+ Manage External Devices	5					.						
+ Manage Itineraries	6			1		1	.	1				
+ Modify User Profile	7		1	1				.				
+ Query Data	8						1		.			
+ Register Views	9			1						.		
+ Save Data	10		1				1	1			.	
+ Show Itineraries	11			1			1					.

Dependency Structure Matrix (DSM) showing the initial responsibilities with their dependencies

*Responsibilities and their dependencies*



# Relating Functional and Quality Attribute Requirements

---

Functional requirements in any system never come without (sometimes implicit) quality attribute requirements.

No quality attribute requirement can exist without a function it is attached to.

Requires the definition of a relationship between functional and quality attribute requirements.



# Scenario – Responsibility Mapping

Specify what responsibilities are affected by what scenario.

Scenario \ Responsibility	A	B	C	D	E
• Performance Scenario 1	X		X		
• Performance Scenario 2		X		X	
• Performance Scenario 3			X		
• Performance Scenario 4	X				X
• Modifiability Scenario 1		X			
• Modifiability Scenario 2				X	
• Modifiability Scenario 3			X		X
• Modifiability Scenario 4	X			X	
• Modifiability Scenario 5	X				

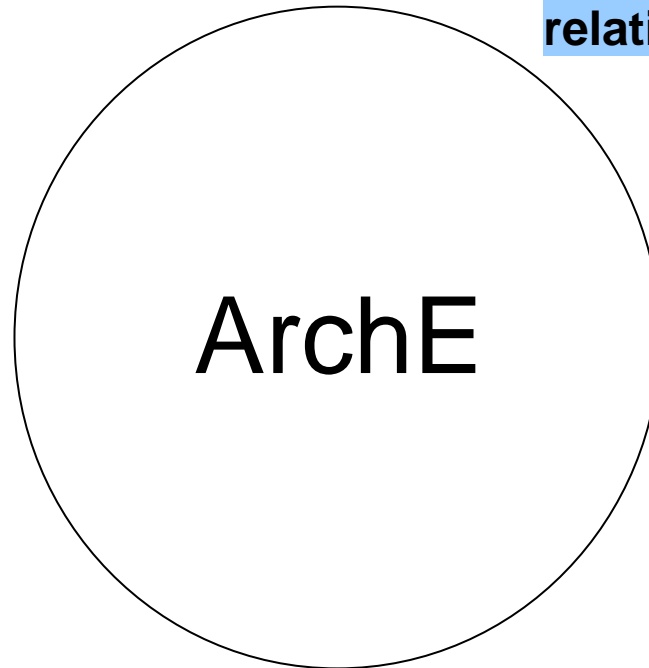


# Requirements Input

Designer specifies functional requirements

Every time this guy shows up we will talk specifically about what ArchE does

At the end, ArchE knows the functional requirements with their relationships.



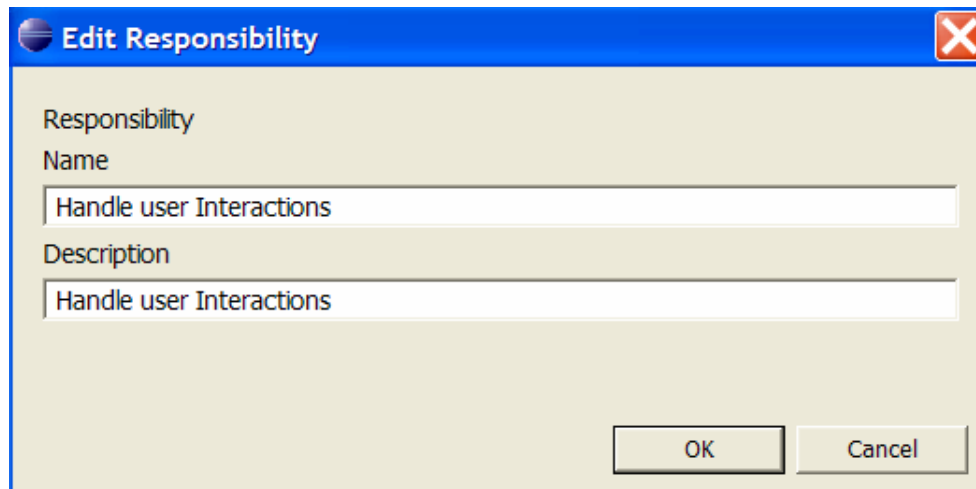
ArchE translates them into responsibilities

Designer specifies dependencies between responsibilities



# Responsibility Entry

---



**Edit Responsibility**

Responsibility  
Name  
Handle user Interactions

Description  
Handle user Interactions

OK Cancel



# Responsibility relationship

R  
Ar  
u.  
se

●●	Adding a new feature requires a change to th...	Modifiability	Add a ne...						The impl...	within 3...	3.5
●●	A new variable is added to the user profile ha...	Modifiability	Add a ne...						it is com...	within 5 ...	5.0
●●	Th driver for a new external device has to be...	Modifiability	Add a ne...			Driver fo...			Device is...	within 10...	10.0
●●	The system has to manage the external devi...	Real-time...	external ...	Periodic	external ...	system	normal		handles t...	under 5 ...	49.0

Scenario-Responsibility Mapping    Function-Responsibility Mapping    Relationships x

Responsibilities or relationship contains:

Parent responsibility	Relations...	Child responsibility	Parameter
Attach to Model	depende...	Register Views	Probability
Create user Profile	depende...	Modify user Profile	Probability
Create user Profile	depende...	Save Data	Probability
Handle user Interactions	depende...	Create user Profile	Probability
Handle user Interactions	depende...	Manage Itinerary	Probability
Handle user Interactions	depende...	Modify user Profile	Probability
Handle user Interactions	depende...	Register Views	Probability
Handle user Interactions	depende...	Show Itinerary	Probability
Manage External Devi...	depende...	Manage Itinerary	Probability i... 0.7
Manage Itinerary	depende...	Query for Data	Probability i... 0.7
Manage Itinerary	depende...	Save Data	Probability i... 0.7
Manage Itinerary	depende...	Show Itinerary	Probability i... 0.7
Manage user Profile	Contains	Create user Profile	Probability i... 0.7
Manage user Profile	Contains	Modify user Profile	Probability i... 0.7
Modify user Profile	depende...	Manage Itinerary	Probability i... 0.7
Modify user Profile	depende...	Save Data	Probability i... 0.7
Query for Data	depende...	Locate Service	Probability i... 0.7

**Edit Relationship** ✖

Responsibility

Relationship

dependency  
 Contains  
 Sequence

OK    Cancel

Parameter



# Requirements Input

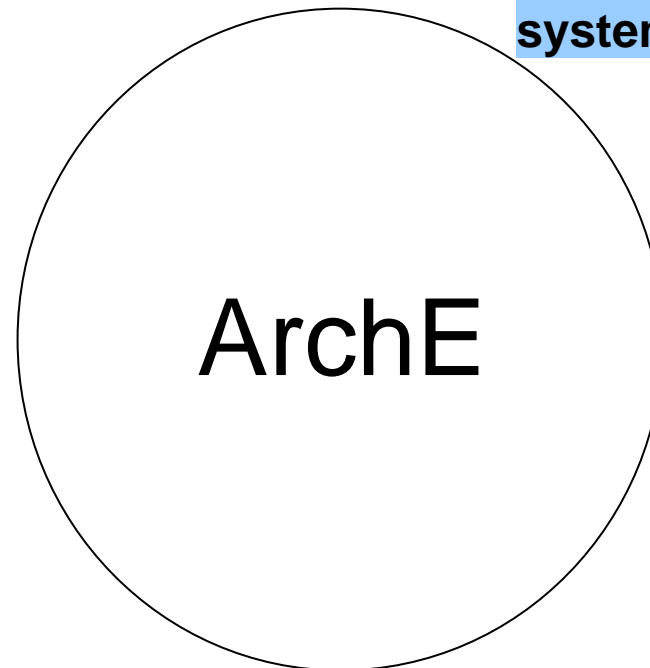
Designer specifies quality attribute scenarios and relates them to responsibilities



ArchE checks the scenarios and may ask for more input

Designer provides more input for each scenario

At the end, ArchE knows the quality attribute requirements for the system being designed.



# Scenario Entry

**Scenario** ✕

**Scenario** S

A scenario is a quality attribute requirement of a system and is described in six parts.

Scenario Text:

Adding a new feature requires a change to the storage format. The implementation changes required for the new format are completed in 3.5 days

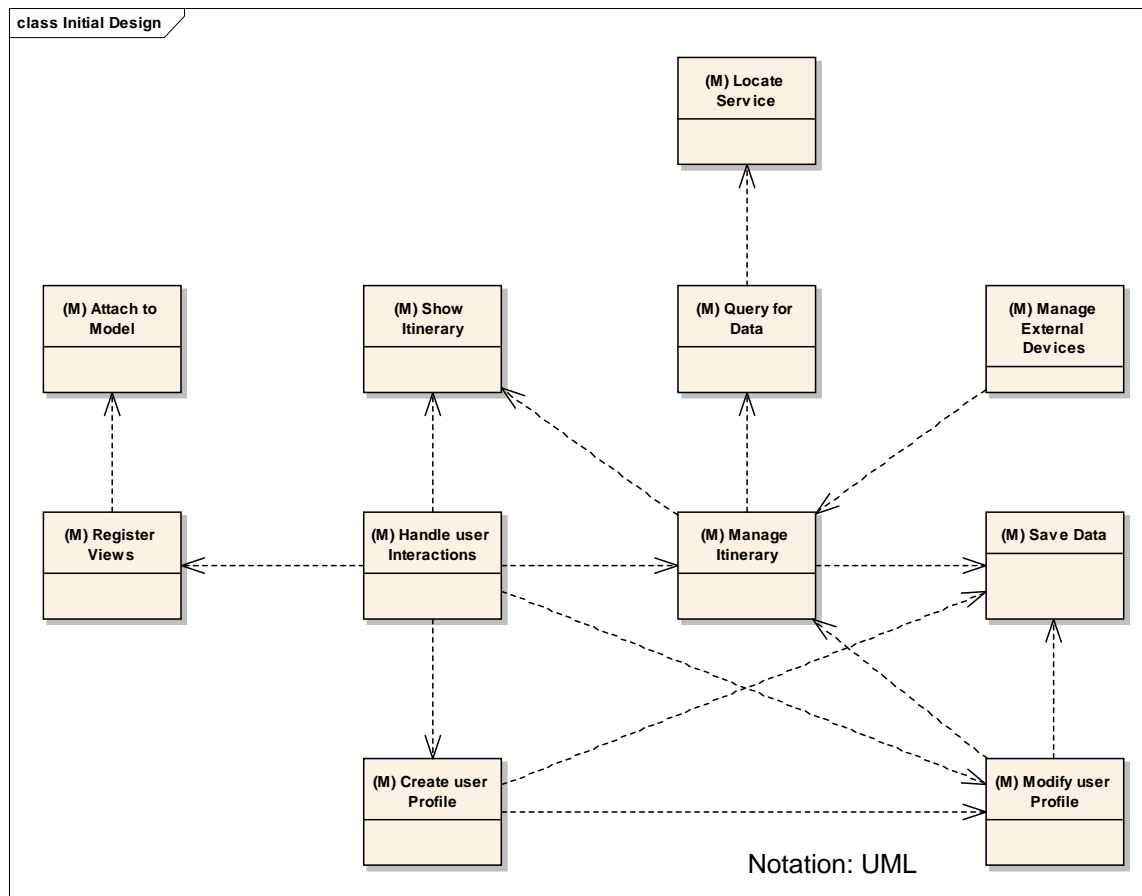
Type:

Six Parts

	Text	Type	Unit	Value
Stimulus:	<input type="text" value="Add a new feature"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Source of stimulus:	<input type="text"/>	<input type="text" value="End user"/>	<input type="text"/>	<input type="text"/>
Environment:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Artifact:	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Response:	<input type="text" value="The implementation is complete"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Response measure:	<input type="text" value="within 3.5 days"/>	<input type="text" value="Cost Constraint"/>	<input type="text" value="Days"/>	<input type="text" value="3.5"/>



# Example – Initial Design



*Initial Design*

UML Class Diagram when assigning each responsibility to its own module (class)

That is what ArchE does if designer did not specify anything different



# Scenario responsibility mapping

The screenshot shows the ArchE Eclipse SDK interface. The main window displays a table of scenarios with the following data:

Description	Scenario...	Stimulus	Stimulus...	Source	Artifact	Environ...	Response	Measure	Value
The user asks to save the current data on th...	Real-time...	save data	Sporadic	user	system	normal c...	data is sa...	in under ...	10000.0
The user asks to show the itinerary under no...	Real-time...	show itin...	Stochastic	user	system	Normal c...	itinerary i...	in under ...	5000.0
The user modifies their profile under normal c...	Real-time...	modify p...	Stochastic	user	system	normal c...	profile is ...	in under ...	5000.0
A view wishes to attach to the model under n...	Real-time...	attach to...	Sporadic	view	system	normal c...	view is at...	in under ...	1000.0
Adding a new feature requires a change to th...	Modifiability	Add a ne...					The impl...	within 3...	3.5
A new variable is added to the user profile ha...	Modifiability	Add a ne...					it is com...	within 5 ...	5.0
Th driver for a new external device has to be...	Modifiability	Add a ne...			Driver fo...		Device is...	within 10...	10.0
The system has to manage the external devi...	Real-time...	external ...	Periodic	external ...	system	normal	handles t...	under 5 ...	49.0

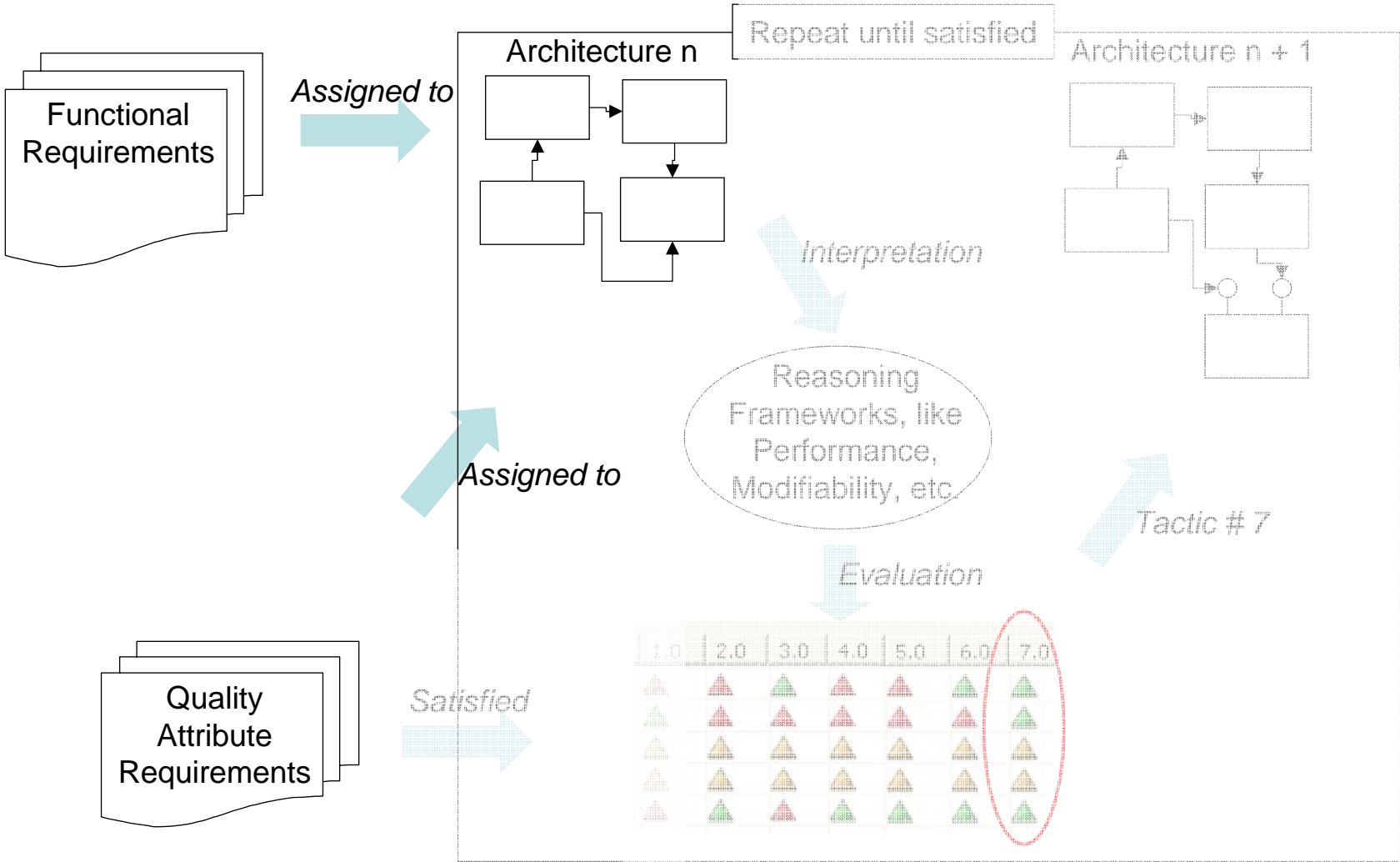
The 'Edit Scenario-Responsibility Mapping' dialog box shows the following configuration:

- Scenario: Adding a new feature requires a change to the storage format. The impleme...
- Responsibility: Save Data

Buttons: OK, Cancel



# The Principles of Architecture Design





# Managing the Reality:

*Reasoning Frameworks Part 1:  
Creating Quality Attribute Models*



# ArchE Supports Multiple Quality Attributes

---

*A Reasoning Framework encapsulates the knowledge needed to enable ArchE (or a designer) to reason about a specific quality attribute*

Allows for extension of quality attribute knowledge within ArchE by plugging in a new reasoning framework

Reduces interactions (dependencies) among quality attributes.

One of the research questions is the extent to which interaction among quality attributes can be reduced. - will return to this idea when we discuss tradeoffs.



# Reasoning Frameworks within ArchE – 1

---

## A reasoning framework within ArchE

1. Translates from architecture description to quality attribute model – we call this “Interpretation”
2. Evaluates quality attribute scenarios in terms of the model – we call this “Evaluation”
3. Proposes tactics to improve architecture.

## Two inputs into a reasoning framework within ArchE

1. Current architecture
2. Relevant quality attribute scenarios

## Outputs:

1. Evaluation of current architecture with respect to the quality attribute scenarios
2. List of potential tactics to improve the architecture if at least one scenario is currently unmet





# Reasoning Frameworks within ArchE – 2

---

Requires a clear definition of the architectural elements, relations, and properties that can influence a quality attribute.

- The Interpretation extracts this information from an architecture and creates a model from it

Requires the existence of a “Formula” to do calculations with the model to provide some information about the fulfillment of the quality attribute

- That is what the “Evaluation” does

Requires a clear definition of possible changes to the architecture to make it better fulfill the quality attribute

- This is what “tactics” are for



# Example: Modifiability Reasoning Framework – 1

---

The modifiability of an architecture depends on the assignment of functionality to modules and the dependencies between the modules. The modifiability is measured in cost (effort) of change.

Therefore the following information must be available:

- Responsibility graph
  - Have dependencies
  - Can be decomposed
- Responsibility properties
  - Cost of change
- Dependency (between responsibilities) properties
  - Strength of coupling
- Responsibilities are assigned to modules



# Example: Modifiability Reasoning Framework – 2

---

Assigning costs of change to each responsibility is job of architect. There is no way ArchE can know initial values.

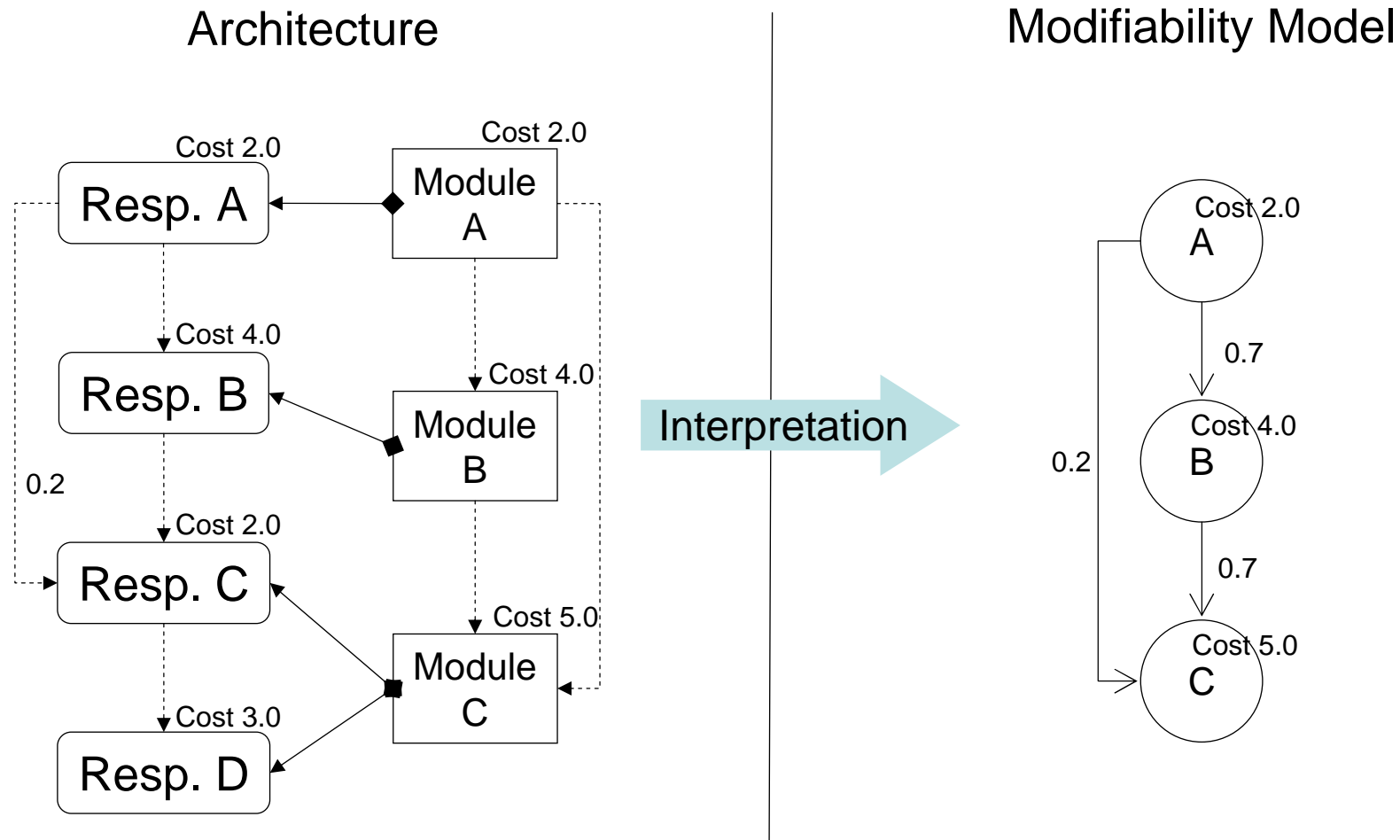
If architect did not assign a strength of coupling for dependencies between responsibilities, then ArchE assumes a default probability (0.7).

Constructing model from architecture description is easy because of the form of the architecture description.

- Each module has a cost of change which is the sum of the cost of change of the assigned responsibilities
- Each module, that is not decomposed, becomes a node in the model
- Each node has a cost of change that is the cost of change of the module
- The dependencies between responsibilities directly determine the dependencies between modules
- Module dependencies become the arcs in the model that connect the nodes
- Each module dependency has a strength of coupling, which is assigned to the arcs in the model



# Example: Modifiability Reasoning Framework – 3



# Example: Computing Predicted Cost Of Change - 1

---

Begin with a scenario that gives requirement

Every scenario is tied to responsibilities that are impacted by the scenario  
(architect does this)

Determines the modules affected by the scenarios

Compute expected cost of change for that scenario as the sum of the expected cost of change affected modules.

Now need to worry about ripples.

The average cost of changing the neighboring module (B) of module (A) equals the cost of change for module (B) times the strength of coupling between module (A) and (B).

Add all the costs and you have the average cost of change for the scenario.



# Validity of cost model

---

Cost model has not been validated

Has some plausibility – it is based on standard concepts of coupling and cohesion

It has the properties that were assumed for derivation of tactics

Has the most impact on ArchE's actions when in multi-step mode



# Performance theory

---

The ArchE performance reasoning framework is based on Rate Monotonic Analysis (suitable for reasoning about real time deadlines).

The theory used in ArchE has the following assumptions:

- Single processor
- Basic computational unit is a *task*
- There may be resources shared among tasks
- Tasks are independent except for explicit reliance on shared resources
- Only one task can use a shared resource at a given point in time
- Processor scheduling priorities are given by the task order – i.e. task 1 is highest priority, task 2 is second, etc



# Example: Performance Reasoning Framework – 1

---

The performance of an architecture depends on the assignment of functionality to tasks. One of the typical measures for performance is Latency – the time it take to finish a task.

Therefore the following information must be available:

- Performance scenarios
  - Have period
- Scenario to responsibility assignment
- Responsibility properties
  - Execution time





# Example: Performance Reasoning Framework – 2

---

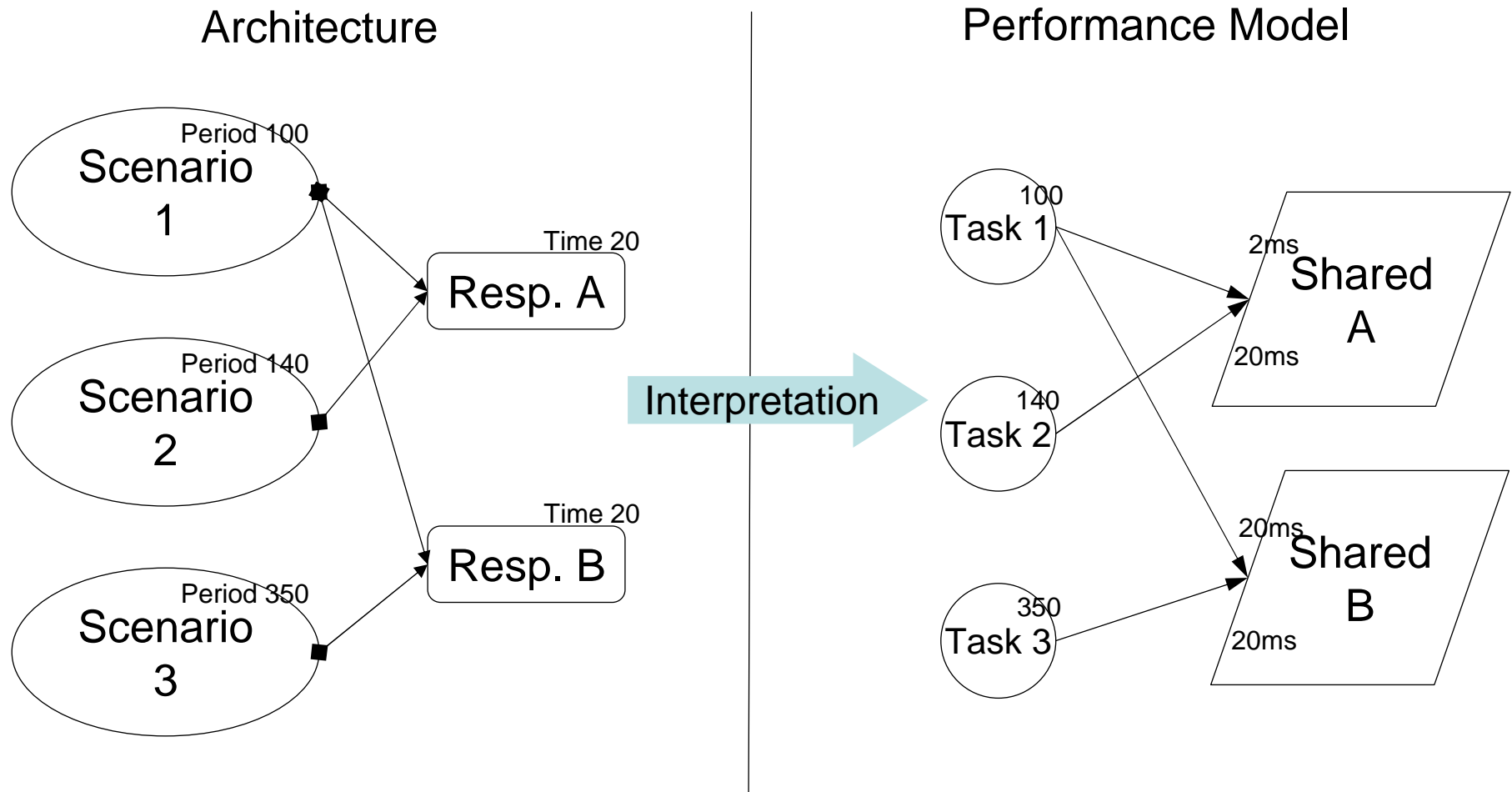
Assigning each responsibility an execution time is job of architect. There is no way ArchE can know initial values.

Constructing model from architecture description is as follows:

- Each performance scenario becomes a task
- The period specified for the scenario becomes the period of this task
- Each responsibility has an execution time
- Responsibilities assigned to a scenario become responsibilities assigned to the task
- Responsibilities not assigned to a performance scenario are assigned to an additional, low priority task (background task)
- Shared responsibilities become shared resources
- A shared resource has an execution time for each task that uses this resource



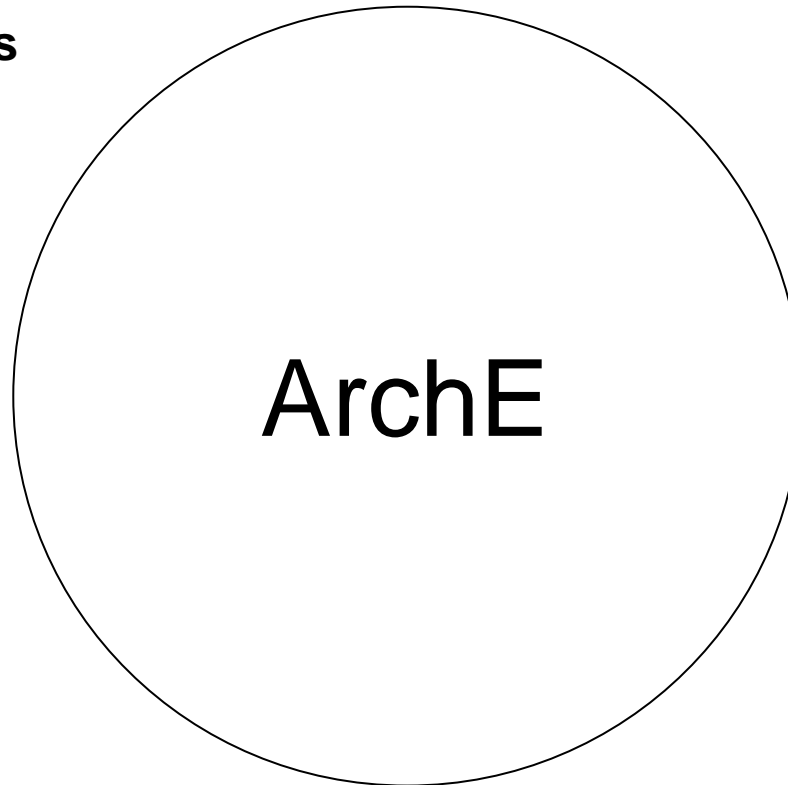
# Example: Performance Reasoning Framework – 3



# ArchE Completes Initial Information

ArchE has scenarios and initial responsibilities

Evaluate for performance and determine missing information



Evaluate for modifiability and determine missing information

Designer provides missing information



# Edit responsibility properties

ArchE - ModifiabilityReasoningFrameworks-rules.clp - Eclipse SDK

File Edit Source Navigate Search Project Run Window Help

Scenarios TrafficLight Responsibilities Functions

Name contains:

Name	Cost of change (\$)	Exec.time (ms)	Level of encapsulation	MutualExclusion	Prepared for change?	Probability incoming (%)	Probability outgoing (%)
Attach to Model	1.0	100.0		FALSE			
Create user Profile	1.0	150.0		FALSE			
Handle user Interactions	4.0	500.0		FALSE			
Locate Service	1.0	900.0		FALSE			
Manage External Devices	2.0	22.5		TRUE			
Manage Itinerary	3.0	600.0		FALSE			
Manage user Profile							
Modify user Profile	1.0	150.0		FALSE			
Query for Data	1.0	700.0		FALSE			
Register Views	1.0	100.0		FALSE			
Save Data	1.0	500.0		FALSE			
Show Itinerary	2.0	450.0		FALSE			

Scenario-Responsibility Mapping Function-Responsibility Mapping Relationships

Responsibilities or relationship contains:

Parent responsibility	Relations...	Child responsibility	Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
Attach to Model	depende...	Register Views	Probability i...	0.7	Probability o...	0.7						
Create user Profile	depende...	Modify user Profile	Probability i...	0.7	Probability o...	0.7						
Create user Profile	depende...	Save Data	Probability i...	0.7	Probability o...	0.7						
Handle user Interactions	depende...	Create user Profile	Probability i...	0.7	Probability o...	0.7						
Handle user Interactions	depende...	Manage Itinerary	Probability i...	0.7	Probability o...	0.7						
Handle user Interactions	depende...	Modify user Profile	Probability i...	0.7	Probability o...	0.7						
Handle user Interactions	depende...	Register Views	Probability i...	0.7	Probability o...	0.7						
Handle user Interactions	depende...	Show Itinerary	Probability i...	0.7	Probability o...	0.7						
Manage External Devi...	depende...	Manage Itinerary	Probability i...	0.7	Probability o...	0.7						
Manage Itinerary	depende...	Query for Data	Probability i...	0.7	Probability o...	0.7						
Manage Itinerary	depende...	Save Data	Probability i...	0.7	Probability o...	0.7						
Manage Itinerary	depende...	Show Itinerary	Probability i...	0.7	Probability o...	0.7						
Manage user Profile	Contains	Create user Profile	Probability i...	0.7	Probability o...	0.7						
Manage user Profile	Contains	Modify user Profile	Probability i...	0.7	Probability o...	0.7						
Modify user Profile	depende...	Manage Itinerary	Probability i...	0.7	Probability o...	0.7						
Modify user Profile	depende...	Save Data	Probability i...	0.7	Probability o...	0.7						
Query for Data	depende...	Locate Service	Probability i...	0.7	Probability o...	0.7						

DesignTreeView.java treeconfig.xml Design.clp InitialModifiabilityDesign-rules.clp ModifiabilityReasoningFrameworks-rules.clp

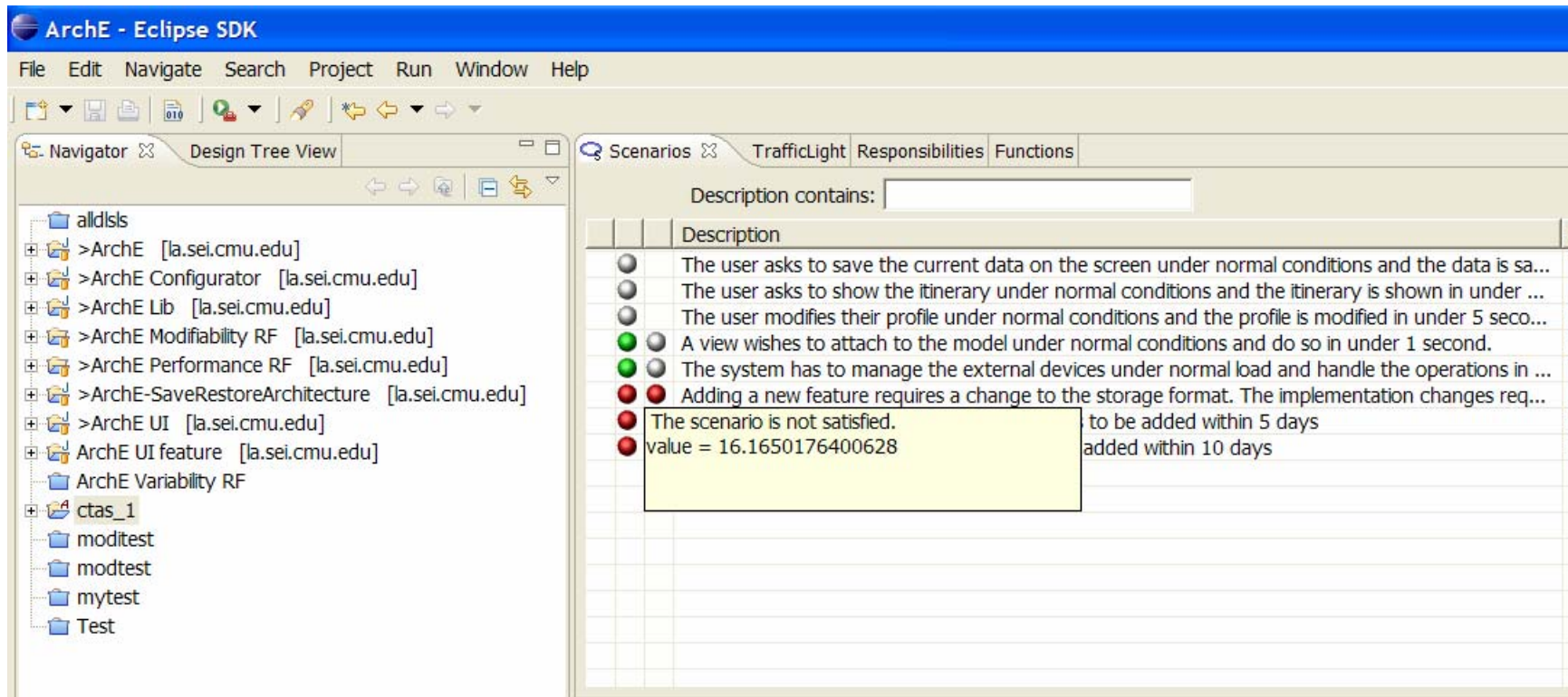
Model Elements Model Relations View Design Elements Design Relations View Questions and Alerts Problems Jess Console Console

Question contains:

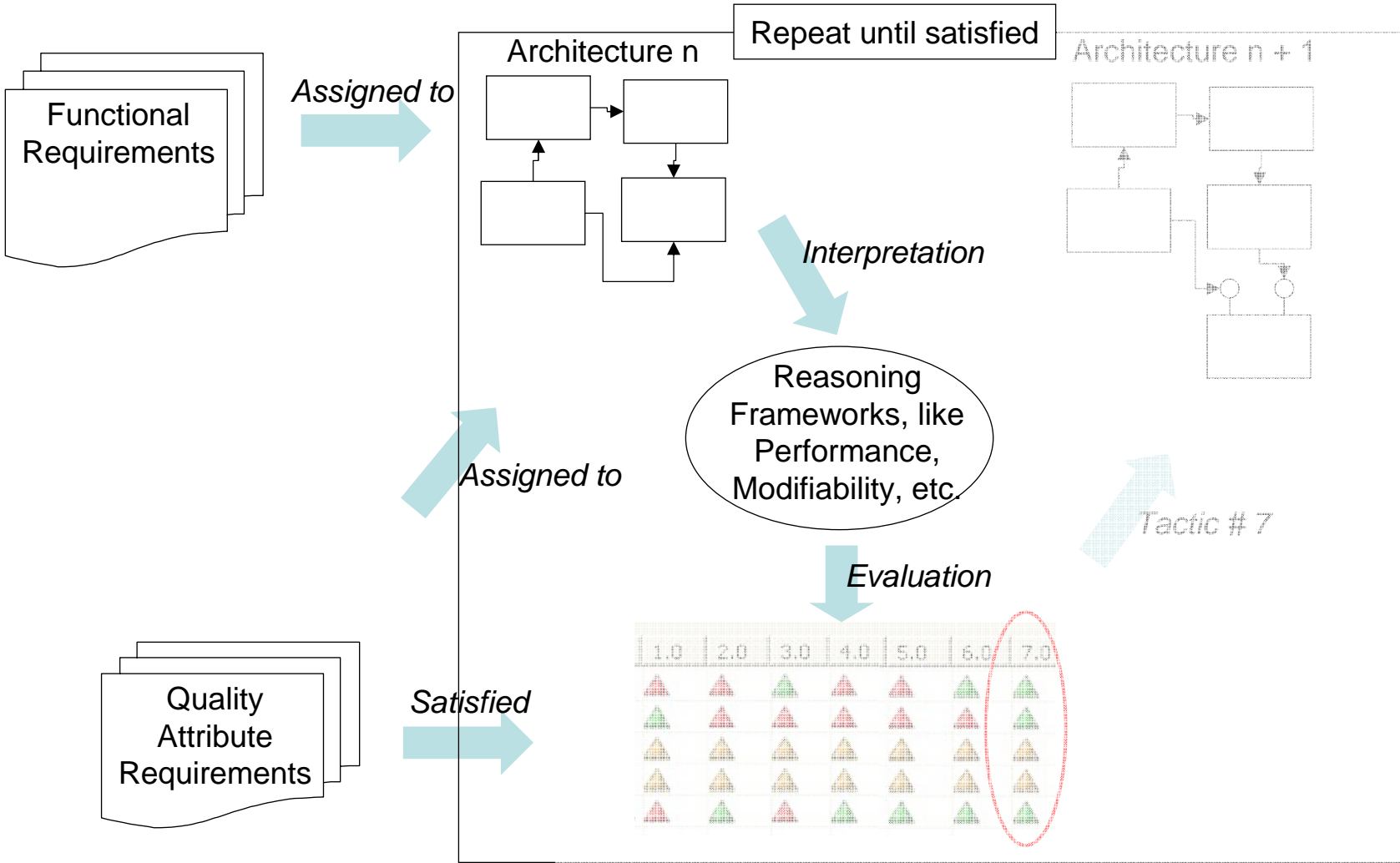


# Scenario Status

Models are created and evaluated as soon as the necessary information is available. Scenarios are marked as satisfied or not.



# The Principles of Architecture Design





# Managing the Alternatives:

*Reasoning Frameworks Part 2:  
Closing in on the Solution*



# Architecture Transformation - Tactics

---

*An architectural tactic is a small transformation of an existing architecture to another one that would better support a specific quality attribute*

A tactic always changes some of the elements and/or properties that are influential in building a quality attribute model

Anything an “Interpretation” uses as input to generate the model can be changed by a tactic. (Knowing what an Interpretation does means knowing what the possible tactics are)

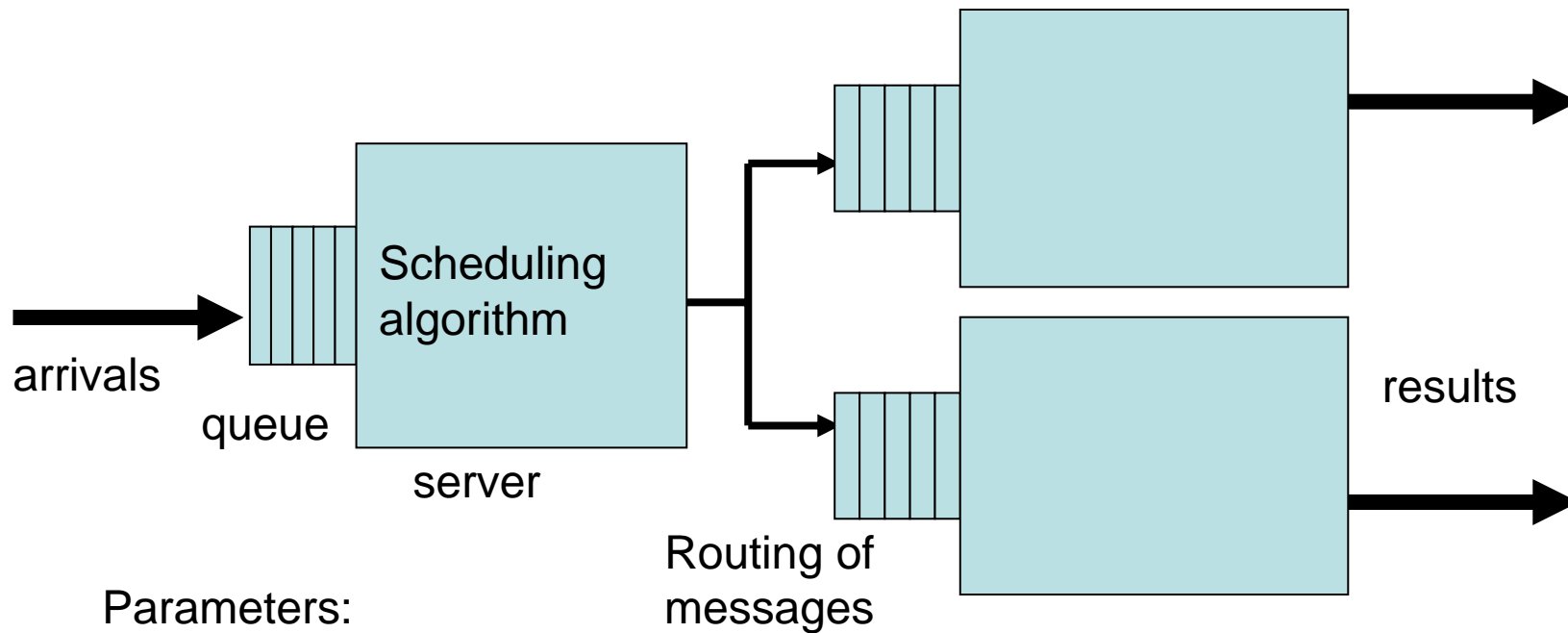
Analyzing a quality attribute model to determine what would produce a better “Evaluation” result guides the selection of possible tactics.

Since a model only contains a fraction of the information about the architecture it is easier to determine possible architecture transformations (tactics).





# Motivating Tactics Through a Queuing Model for Performance



Parameters:

- Arrival rate
- Queuing discipline
- Scheduling algorithm
- Service time
- Topology
- Network bandwidth

Latency (time to compute results) can only be affected by changing one of the parameters



# Performance tactics - 1

---

**Architectural means for controlling the parameters of a performance model**

Arrival rate – restrict access, differential rate/charging structure

Queuing discipline

- FCFS
- Priority queues
- Etc

Network bandwidth – faster networks



# Performance Tactics - 2

---

## Service time

- Increase efficiency of algorithms
- Cut down on overhead (reduce inter-process communication, use thread pools, use pool of DB connections, etc)
- Use faster processor

Scheduling algorithm – round robin, service last interrupt first, etc

Topology – add/delete processors

Routing algorithm – load balancing



# Modifiability Tactics within ArchE

---

The following modifiability tactics are currently implemented in ArchE

- Encapsulation – reduces coupling
- Raising level of abstraction – reduces coupling
- Intermediary – reduces coupling of some dependency
- Splitting responsibility – enables new responsibility to module assignments, changes cost of change
- Localization – changes responsibility to module assignment
- Wrapper – reduces all outgoing couplings

We motivated tactics in terms of the modifiability model but to use tactics in ArchE, more information is needed.

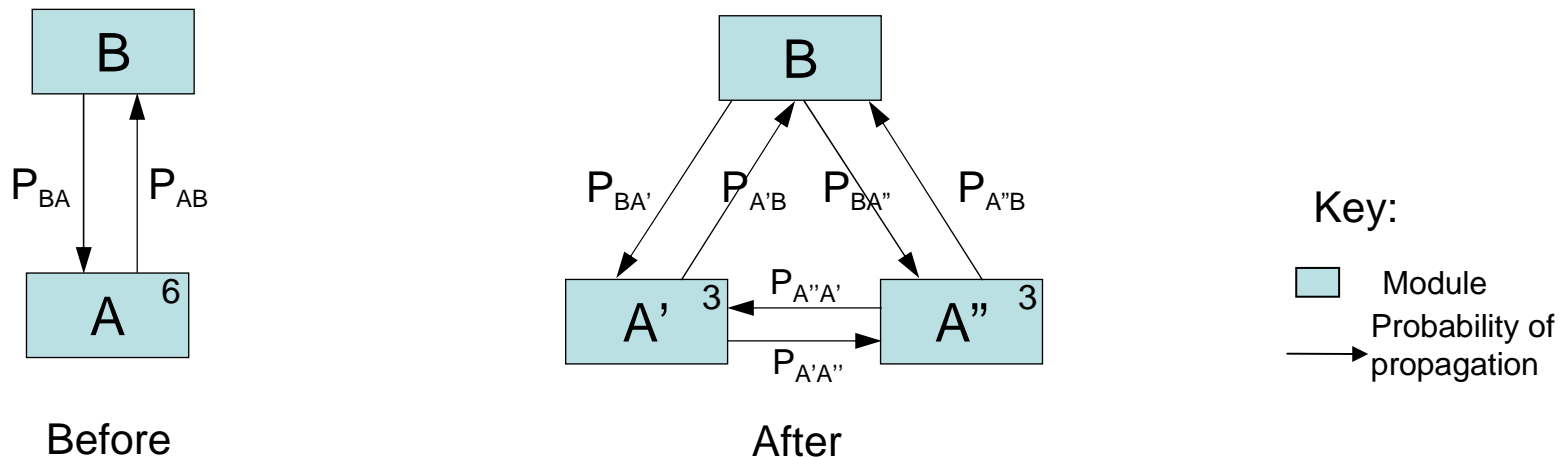


# Example: Splitting Responsibility – 1

If a responsibility is affected by a change then it might be that not everything of that responsibility has to change.

The “Splitting Responsibility” tactics splits a responsibility into two parts, only one of which is affected by a scenario.

Reduces the cost of change.



# Example: Splitting Responsibility – 2

---

What else needs to be specified?

If a responsibility is decomposed into two new responsibilities, the names of the new responsibilities must be specified.

Each new responsibility has a cost of change and probability of propagation to those responsibilities that had dependencies with the responsibility being decomposed.

The two new responsibilities have probabilities of propagation between them.

What does ArchE know?

- ArchE cannot know the names of the new responsibilities.
- ArchE does not know (but has default values – divided by two) for the cost of change of the new responsibilities
- ArchE does not know (but has default values) for the probabilities of propagation.



# Performance tactics implemented in ArchE

---

## Requirements tactics:

- Increase the period
- Increase the deadline

## Scheduling tactics:

- Theory says that scheduling the task with the shortest deadline first is optimal.

## Execution time tactics

- Reduce execution time

## Resource sharing tactics

- Split responsibility – allows executing responsibility with different priorities
- Reduce blocking – split responsibility into portion that uses shared resource and portion that does not.



# Limitations of application of performance tactics

---

Change requirements – ArchE has no means to know to do this (in single step mode)

Reduce execution time – ArchE tests various levels of reduction. Architect (or other means) must determine feasibility

Split responsibility – new names, execution times must be assigned, and new blocking times must be assigned

Reduce blocking – new names, execution times must be assigned, and new blocking times must be assigned

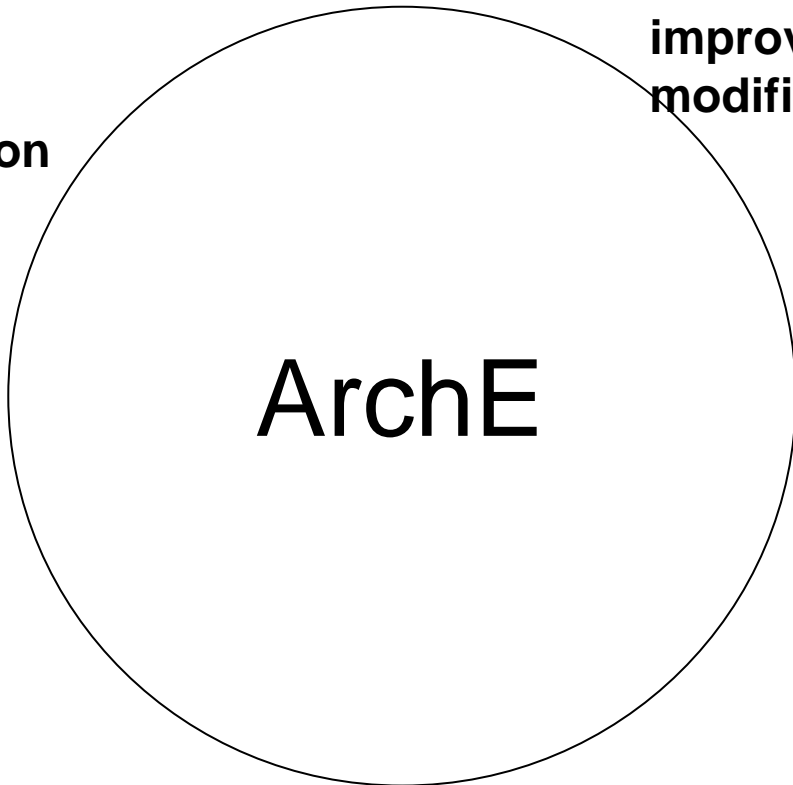




# ArchE's Action

ArchE has scenarios and current architecture with all necessary information

Evaluate for performance and determine possible tactics to improve architecture for performance scenarios



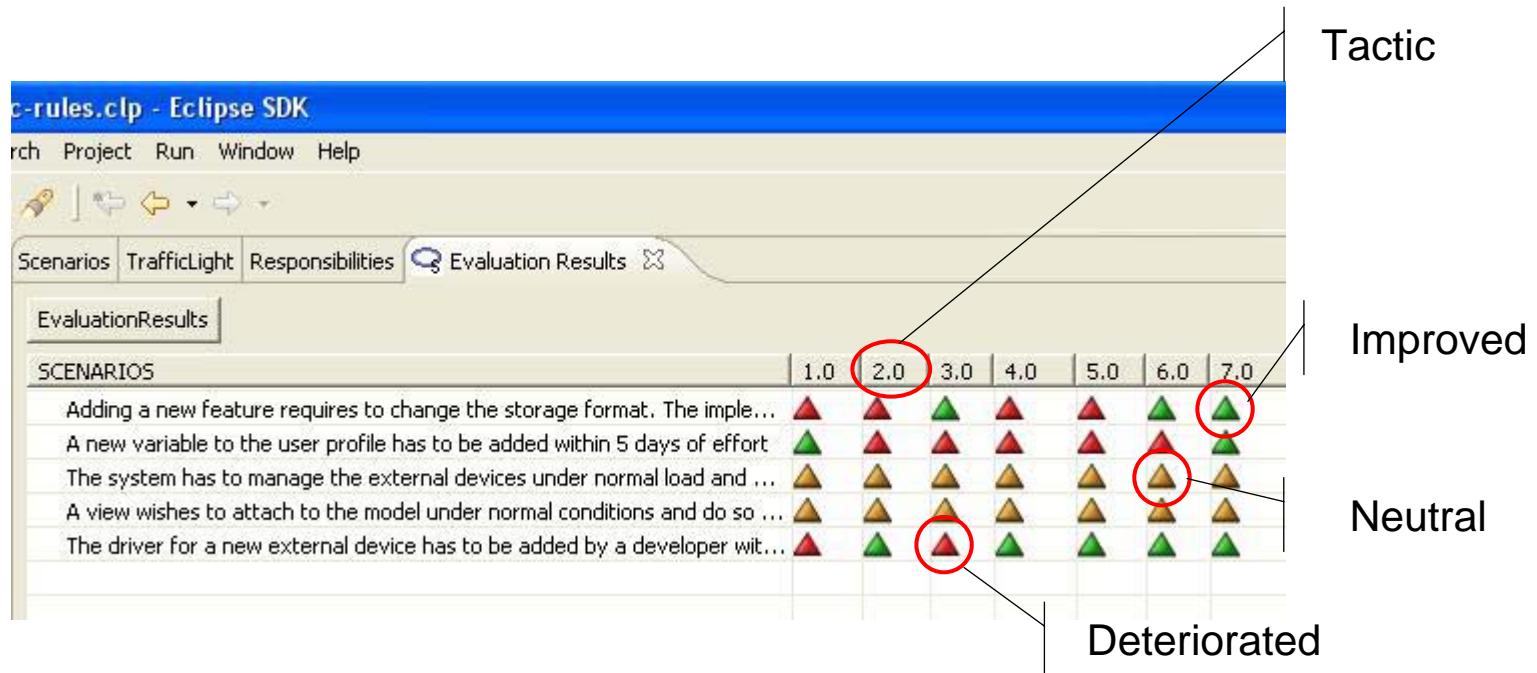
Evaluate for modifiability and determine possible tactics to improve architecture for modifiability scenarios

Display tactics to designer

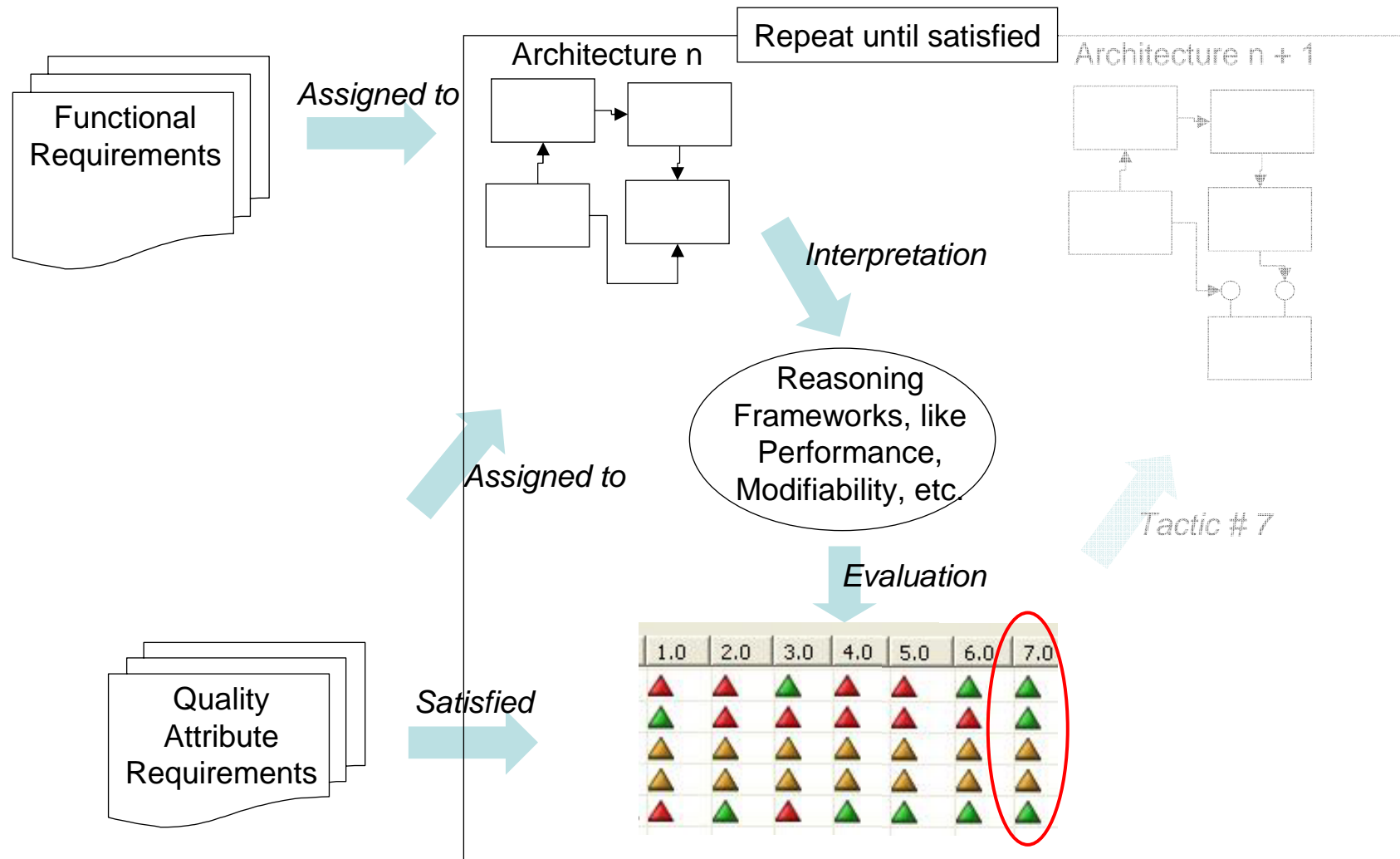


# Evaluation of Tactics

ArchE evaluates suggested tactics and provides information about their influence on the architecture for the designer.



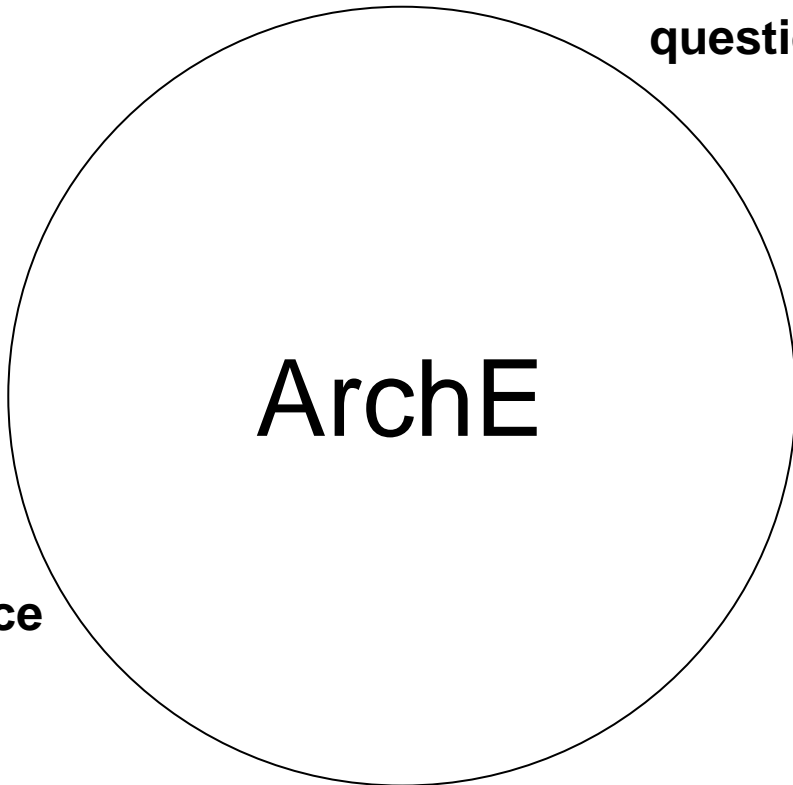
# The Principles of Architecture Design



# ArchE's Action

ArchE uses the quality attribute models to reason about plausible tactics

Evaluate the influence of the plausible tactics on the architecture



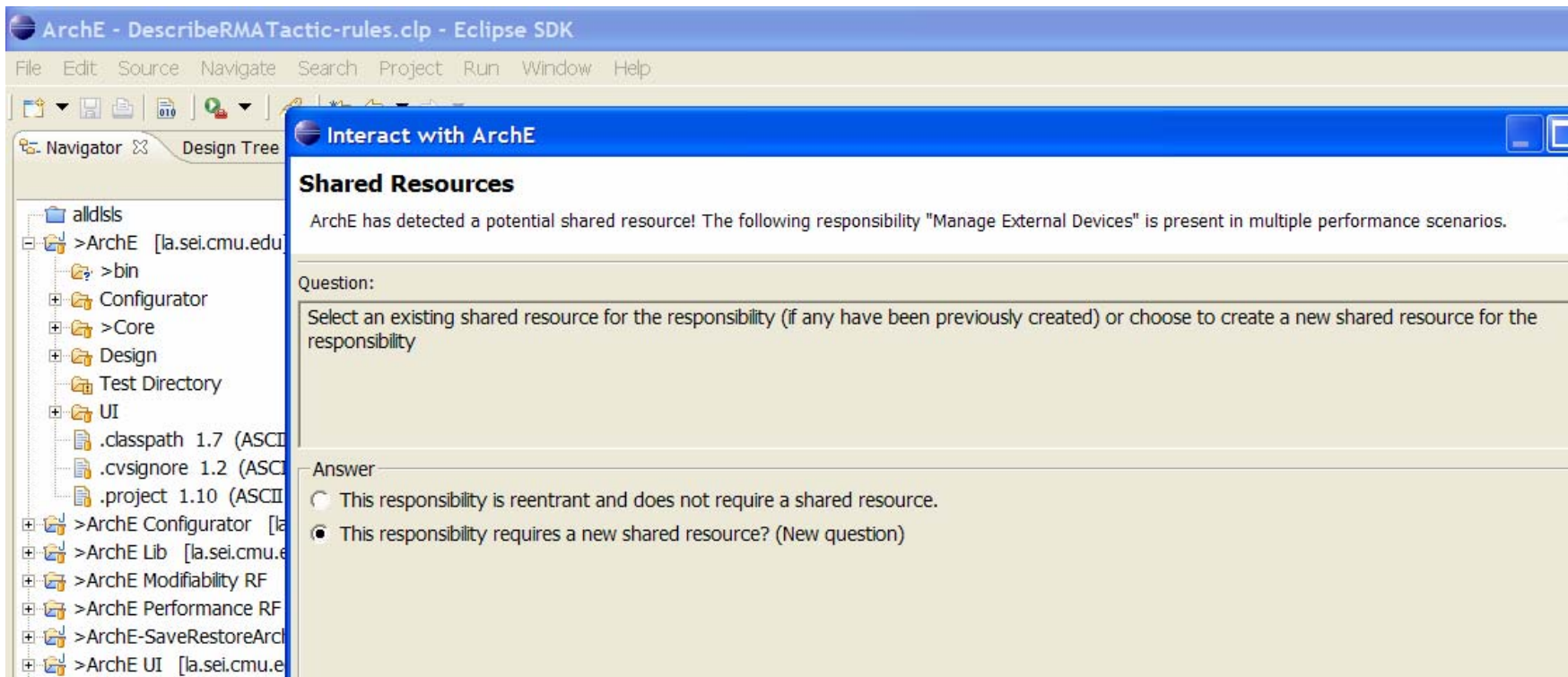
Present the Designer with selected tactics in form of questions

Designer picks a tactic to apply



# Suggested Tactics – Shared Resources

Architect can either accept the suggestion and let ArchE do the transformation or changes the architecture manually.



# Suggested Tactics – Reduced Execution Time

The screenshot shows a software interface with a file explorer on the left and a table on the right. The table has a header 'Description' and one row with the text 'The user asks to save the current data on the screen under normal conditions and the data is sa...'. A blue dialog box titled 'Interact with ArchE' is overlaid on the interface. The dialog box has a title bar 'Interact with ArchE' and a main title 'Confirm ArchE Tactic Suggestion'. The main text in the dialog box reads: 'The scenario "A view wishes to attach to the model under normal conditions and do so in under 1 second." is not satisfied by the current design. Reducing the execution time of the res "960.0" "msec". The task deadline is "1000.0" "msec"'. Below this text is a section labeled 'Question:' with the text: 'Can ArchE reduce the execution time of this responsibility from "900.0" to "810.0" ? If reducing the execution time is possible, please enter the new execution time below.'. At the bottom of the dialog box is a section labeled 'Answer:' with a text input field containing the value '810.0'.



# Influence of Tactics on Other Quality Attributes

---

ArchE immediately evaluates the architecture again after applying a tactic to present possible side effects to the designer.

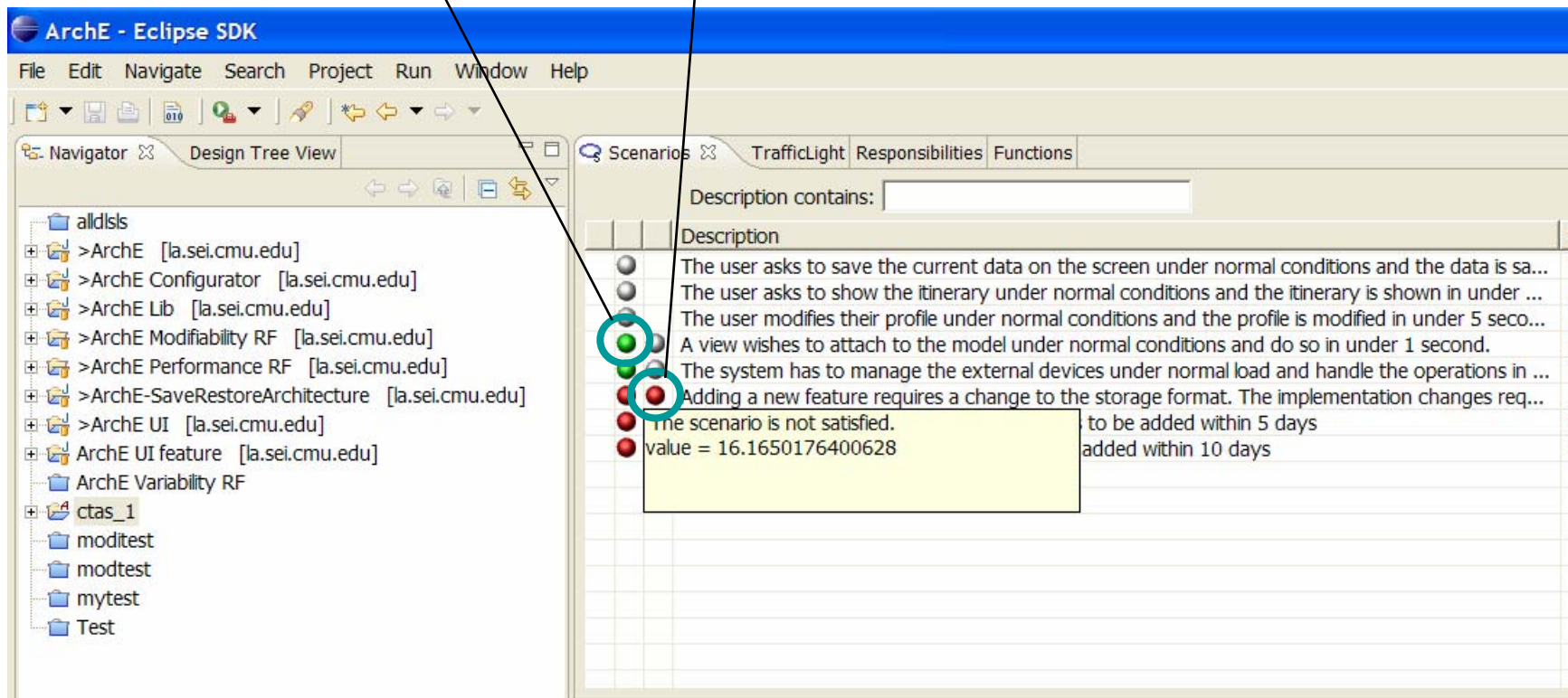
Assume reducing execution time of a responsibility will cause its cost of change to increase since it implies a more complicated algorithm.



# Modifiability Impacted After Reduce Execution Time Tactic

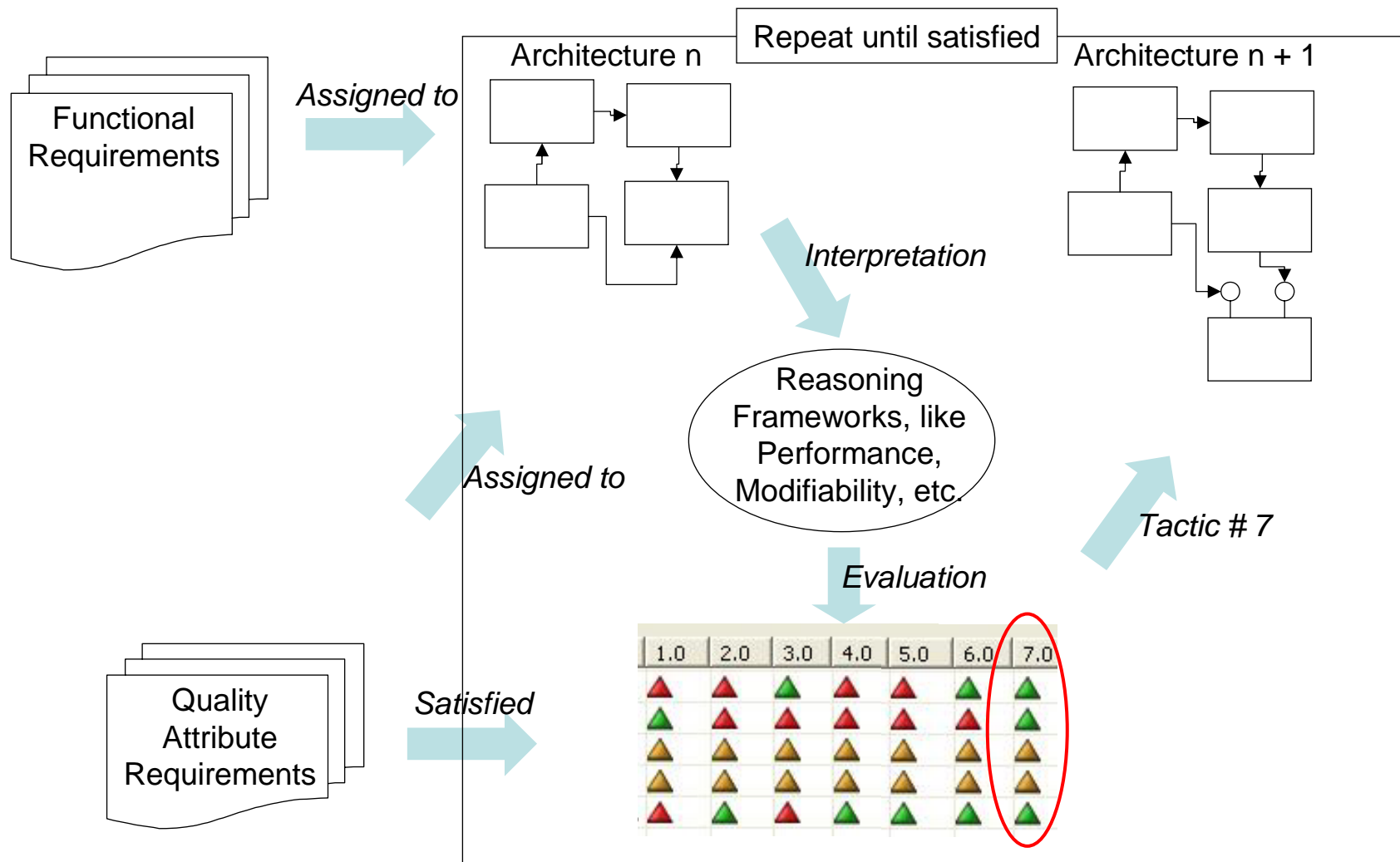
First column depicts if scenario is satisfied or not

Second column depicts change after tactic was applied





# The Principles of Architecture Design



# ArchE Multi-Step Mode – 1

---

So far we presented ArchE in a single step mode.

In this mode ArchE evaluates the current architecture and makes suggestions for the next step.

ArchE can also run in a multi-step mode.

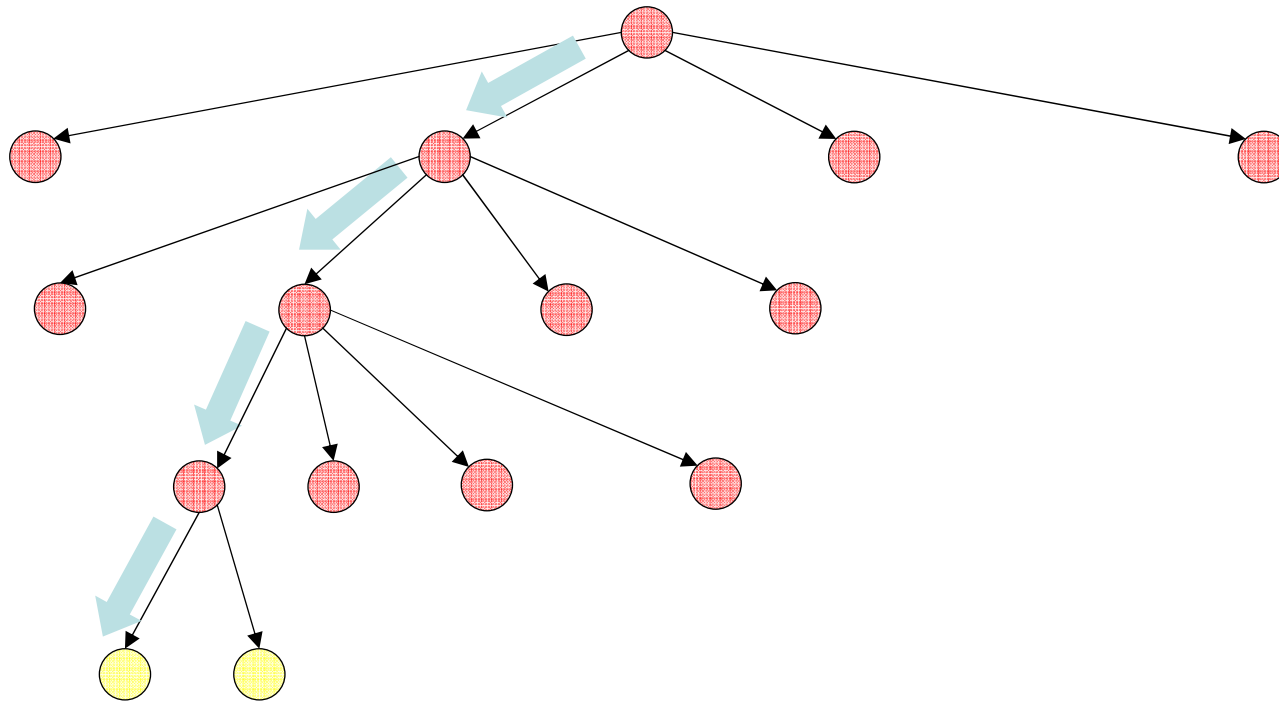
In this mode ArchE automatically chooses one of the suggested tactics, applies it to the architecture and evaluates the consequences.





This is repeated a specified number of times. ... or until a solution is found.

ArchE presents the sequence of chosen tactics to the designer.



# ArchE Multi-Step Mode – 2



-  Unacceptable Architecture
-  Acceptable Architecture
-  Solution!
-  Decision



# Automating Trade off

---

Trade off is giving up one thing of value to achieve something else of value

How is value of architecture measured?

- Ideally against business goals – but we have no method for doing this.
- Can measure how well architecture does against requirements.

Valuing requirements

- Assign value to requirements – quality and functional



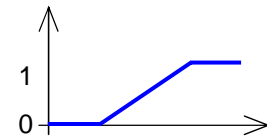
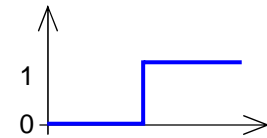
# Utility

---

Utility is a concept dating to the 18<sup>th</sup> century.

For quality attribute requirements

- So far we applied a utility value to scenarios of 0 or 1; 0 equals scenario is not satisfied and 1 means scenario is satisfied
- By scaling to values between 0-1 utility reflects the value of partially satisfied scenarios



ArchE can use the utility value to determine solutions when no solution would be possible otherwise.

Utility is practically an instrument for weakening of requirements.



# Side Effects of Reasoning Frameworks - 1

---

Reasoning frameworks are intended to isolate one quality attribute from another

Parameters for reasoning framework are attributes to responsibilities.

As long as no parameters are shared among reasoning frameworks then they should be independent – correct?



# Side Effects of Reasoning Frameworks - 2

---

Consider performance and modifiability

- Performance parameters are
  - Execution time of a responsibility
  - Scheduling priority of a responsibility
  - Sharing of resources among responsibilities
- Modifiability parameters are
  - Cost of change of a responsibility
  - Probability of propagation among responsibilities

As you can see, nothing is shared between the two reasoning frameworks



# Side Effects of Reasoning Frameworks - 3

---

## Impact of modifiability on performance

- Modifiability can introduce new responsibilities, e.g. introduce an intermediary
- Performance responds to new responsibilities by acquiring or deducing appropriate parameters
- => not a problem so far

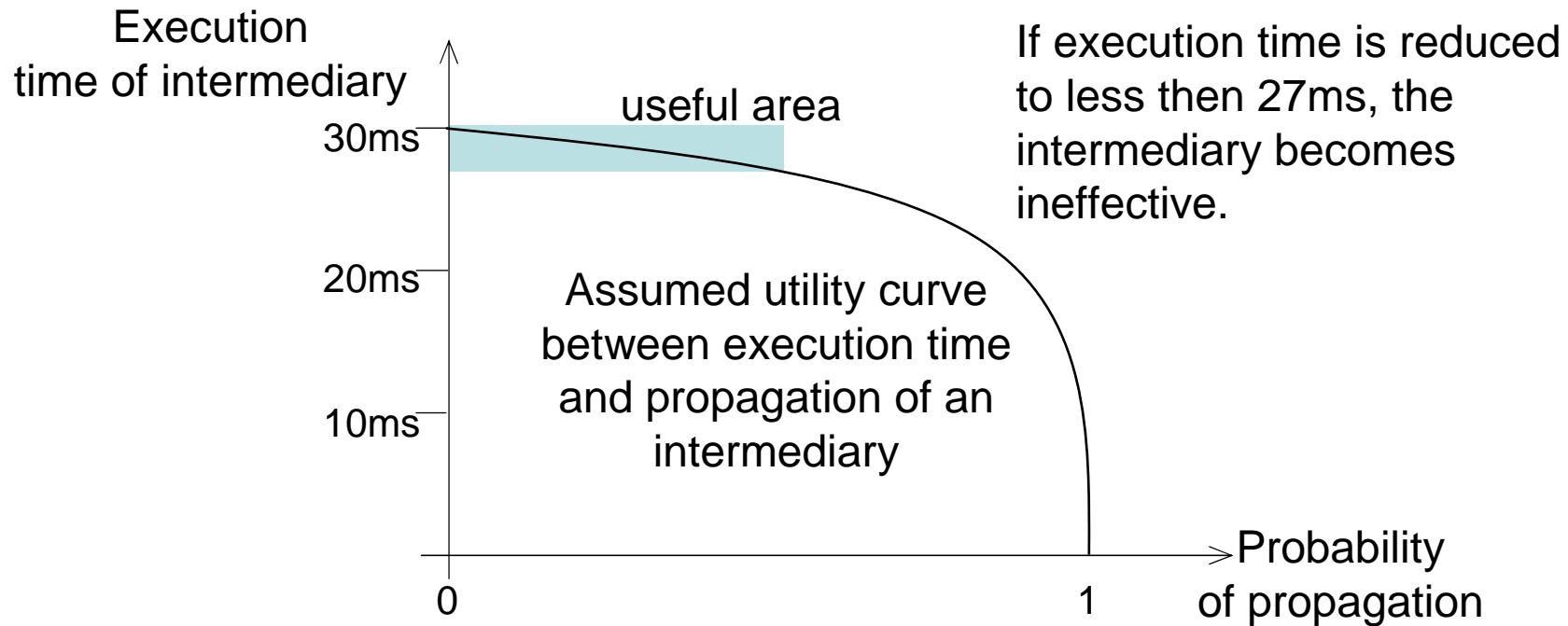
## Continuing ...

- Suppose performance now reduces execution time of the intermediary
- Could impede the function of the intermediary
- => problem since performance needs to understand the purpose of the responsibility and whether reducing execution time will impede the purpose.





# Side Effects of Reasoning Frameworks - 4



Within ArchE, the utility curve determines utility after reducing execution time and there is a separate tradeoff manager whose responsibility is to generate utility curves for potential interactions among reasoning frameworks. (As of today this is hard coded in ArchE)



# Some Final Words about ArchE

---

ArchE is untested with respect to large numbers of scenarios

The Attribute Driven Design (ADD) method focuses on small number of scenarios.

ArchE is useful when used in conjunction with ADD

- Recall from ADD that key steps are
  - Find small number of architectural drivers (scenarios)
  - Design to satisfy this set of architectural drivers
- ArchE can be used in this context



# Plans for ArchE

---

ArchE v2.1 is ready for beta test.

We plan to distribute ArchE freely to two communities:

- Instructors who will use ArchE within a course on software architecture
- Researchers who are interested in developing new reasoning frameworks

Looking for one additional beta test site in each category

Plan on general distribution to these two communities fall 2007.





# Managing the Future:

*Increasing available and codified  
knowledge*



# Extending the Knowledge

---

You want to use ArchE but you have a specific problem to solve that ArchE does not know anything about (Yet!).

Remember:

- If you know what information is required to reason about this problem
- If you know how to distinguish good from bad solutions
- If you know how to change the architecture to make it better

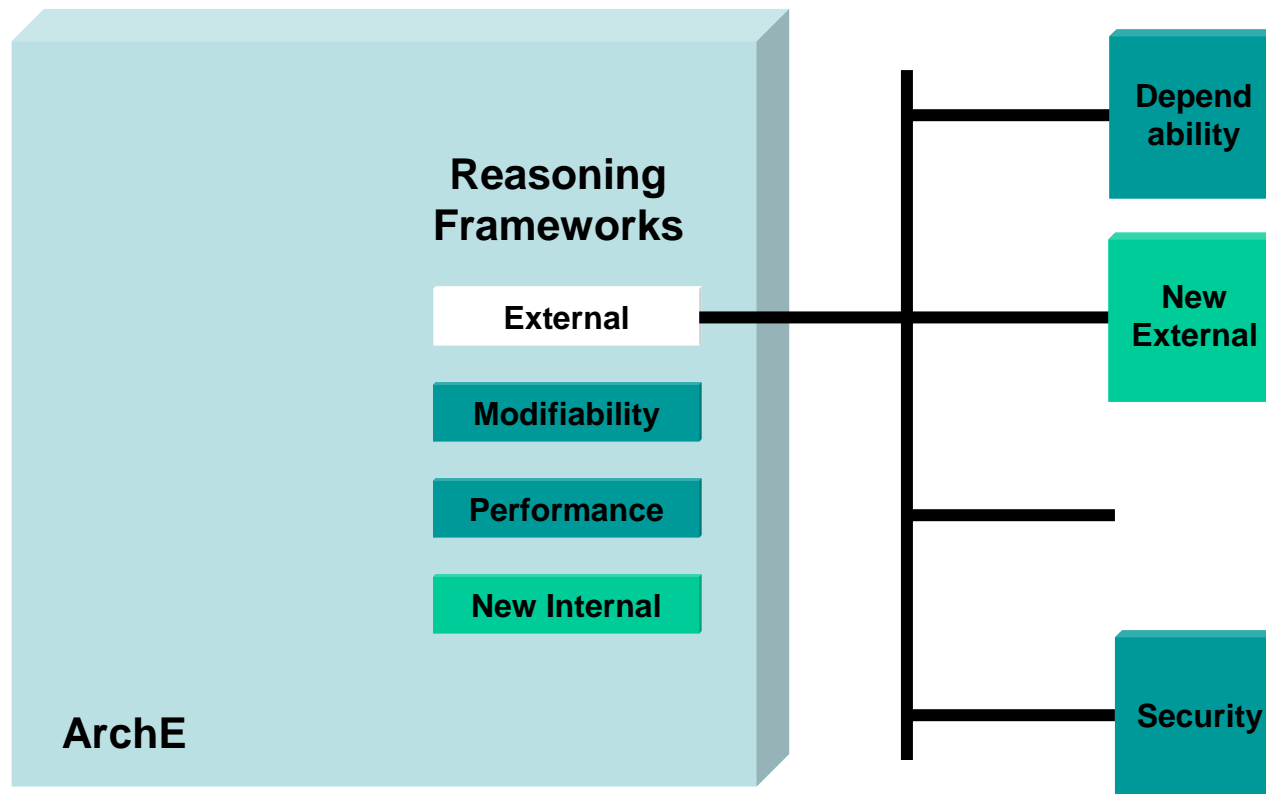
Then you can create your reasoning framework to solve this problem and plug it into ArchE.

... then ArchE automatically checks the side effects your solution may have on other quality attribute that ArchE understands.



# Two methods to extend ArchE

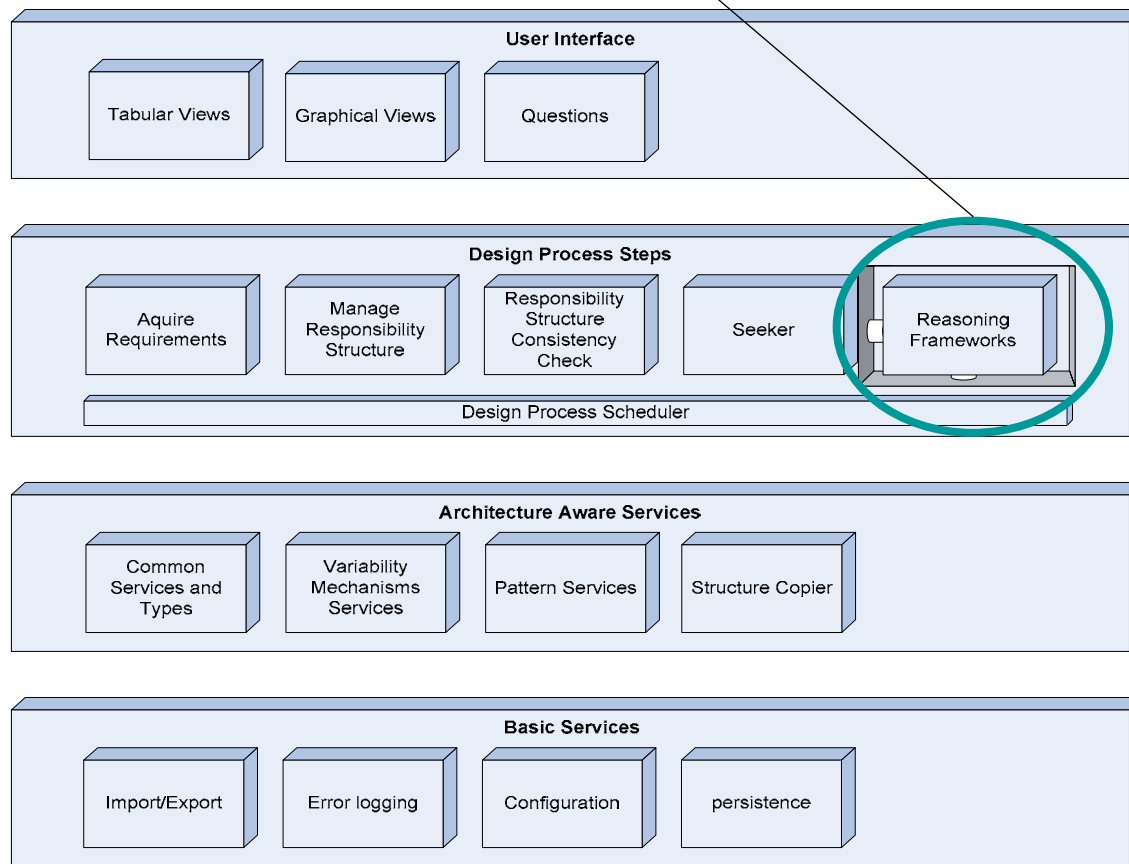
ArchE supports two methods for adding new reasoning frameworks, internal and/or external



# Adding an internal Reasoning Framework

Layered View of ArchE

Reasoning Frameworks are implemented as “plug-ins”



# Development Environment for Internal Reasoning Frameworks

---

ArchE is constructed as an Eclipse plug-in on top of:

- JESS – a rules engine (free to academic institutions)
- Java – generally available
- MySQL Database – also free

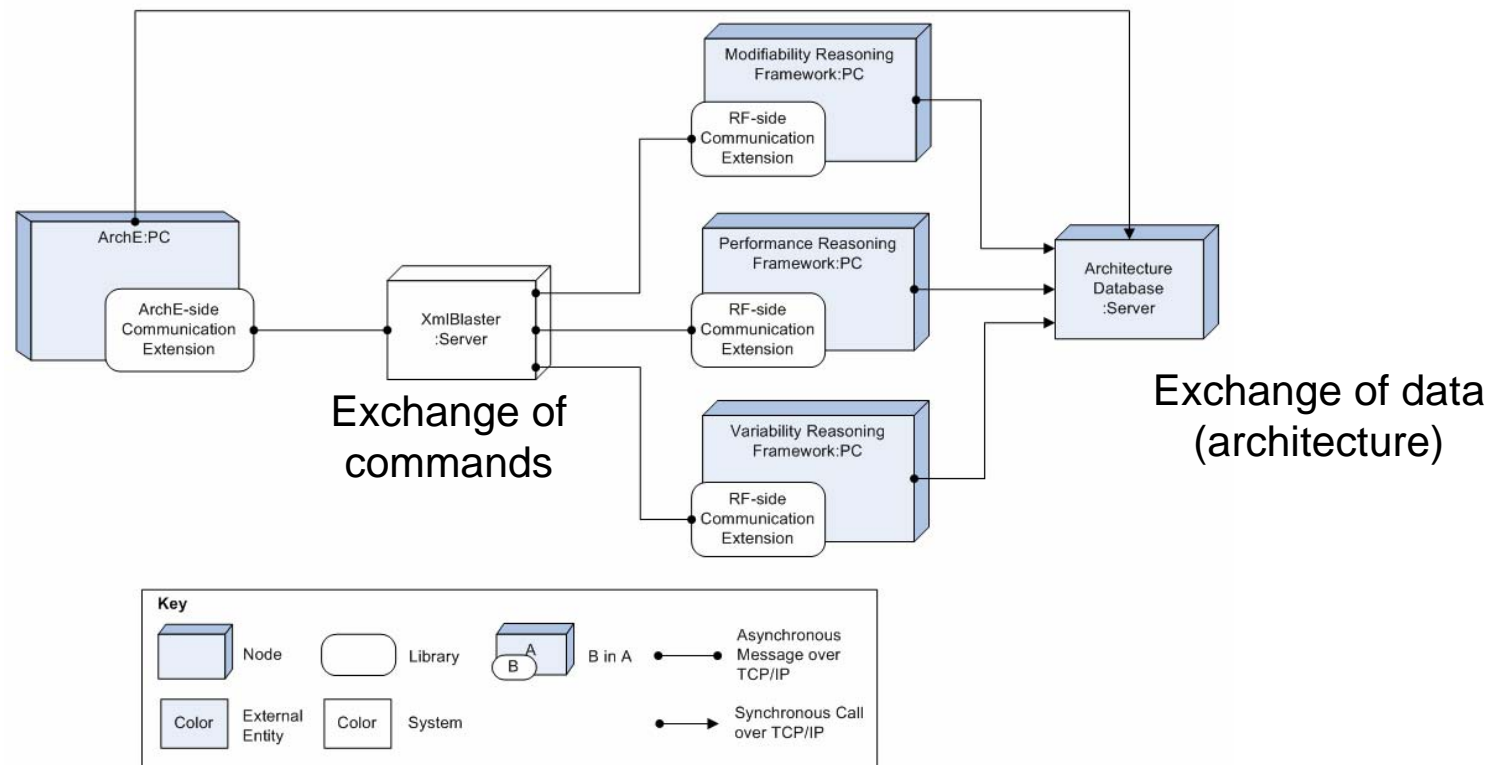
Implementing an internal reasoning framework requires knowledge of this infrastructure.



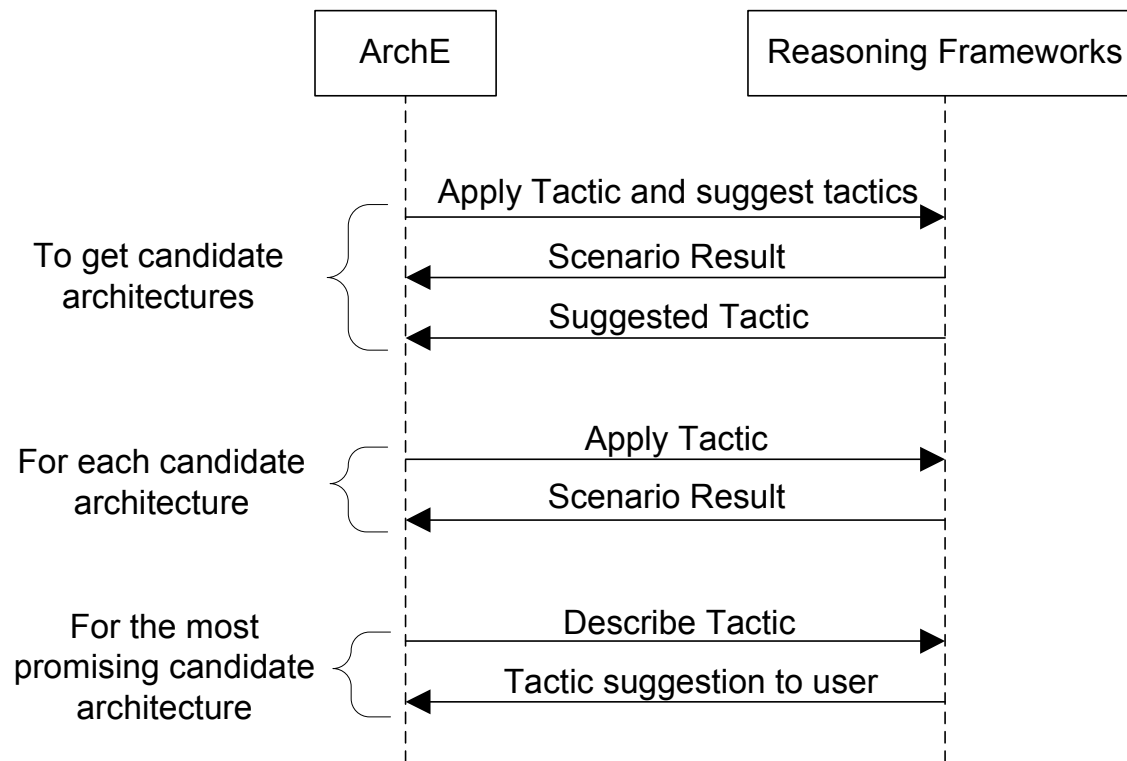


# Adding an external Reasoning Framework

A reasoning framework can be implemented in any language on any type of system and can be connected to an ArchE instance via an XML interface over the net.



# Typical Sequence Between ArchE and Reasoning Framework



# Thank You

---

Questions??



**Software Engineering Institute**

**Carnegie Mellon**

The Architect and ArchE  
Bachmann, Bass, Bianco 03/26/07  
© 2007 Carnegie Mellon University

91



**Software Engineering Institute**

**Carnegie Mellon**