Quality Attribute Modeling and Analysis

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

04-11-2018

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Outline

Modeling Architectures to Enable Quality Attribute Analysis

Quality Attribute Checklists

Thought Experiments and Back-of-the-Envelope Analysis

Experiments, Simulations, and Prototypes

Analysis at Different Stages of the Life Cycle

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

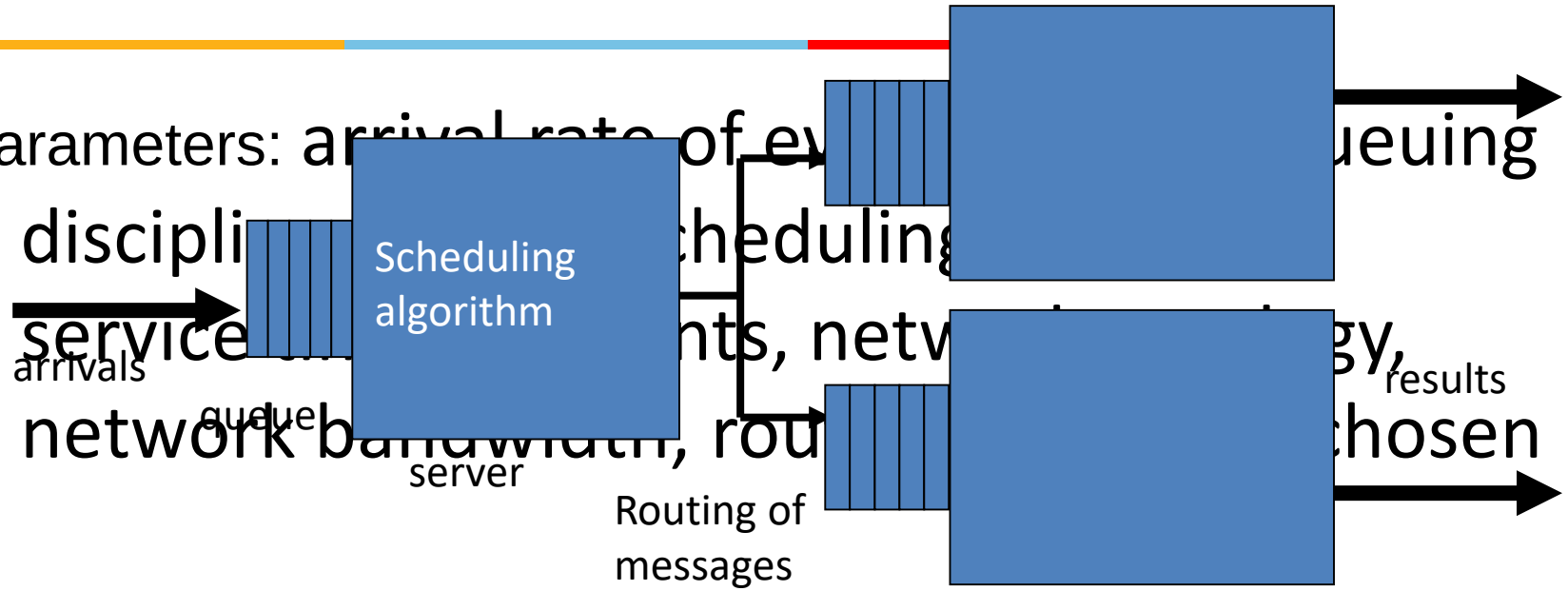
Modeling Architectures to Enable Quality Attribute Analysis

Some quality attributes, most notably performance and availability, have well-understood, time-tested analytic models that can be used to assist in an analysis.

By *analytic model*, we mean one that supports quantitative analysis. Let us first consider performance.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

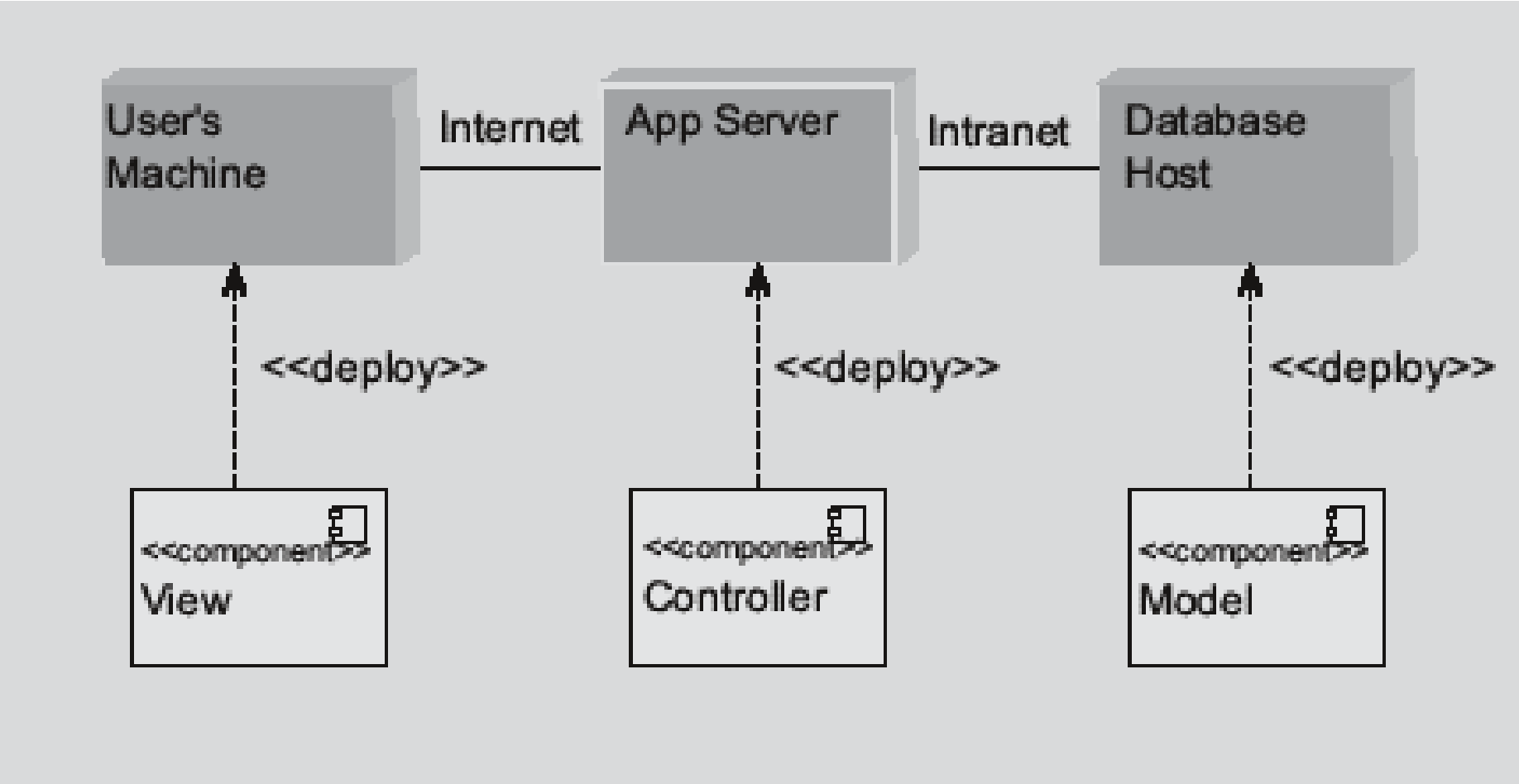
Performance Models



Parameters: arrival rate of ev… queuing discipli… scheduling service … nts, netw… gy, network bandwidth, rou… chosen

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

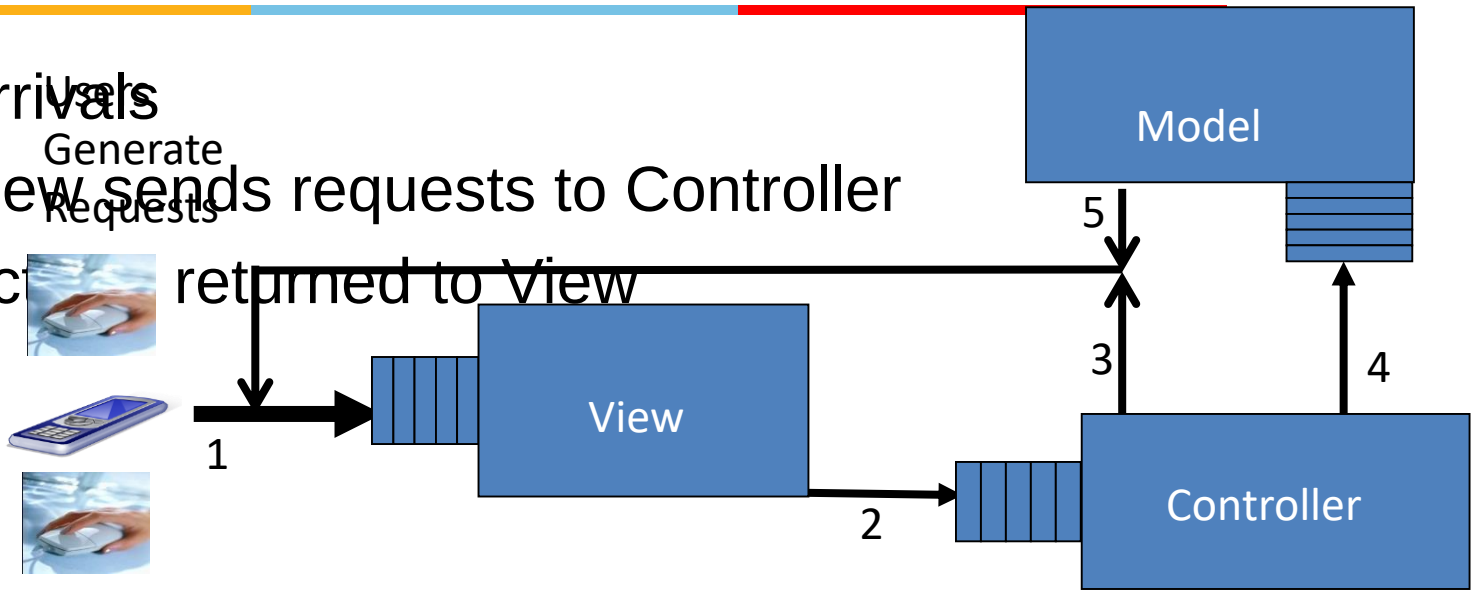
Allocation Model for MVC



© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Queuing Model for MVC

1. Arrivals
2. View sends requests to Controller
3. Actions returned to View



4. Actions returned to model
5. Model sends actions to View

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Parameters

To solve a queuing model for MVC performance, the following parameters must be known or estimated:

- The frequency of arrivals from outside the system
- The queuing discipline used at the view queue
- The time to process a message within the view
- The number and size of messages that the view sends to the controller
- The bandwidth of the network that connects the view and the controller
- The queuing discipline used by the controller
- The time to process a message within the controller
- The number and size of messages that the controller sends back to the view
- The bandwidth of the network from the controller to the view
- The number and size of messages that the controller sends to the model
- The queuing discipline used by the model
- The time to process a message within the model
- The number and size of messages the model sends to the view
- The bandwidth of the network connecting the model and the view

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Cost/benefit of Performance Modeling

Cost: determining the parameters previously mentioned

Benefit: estimate of the latency

The more accurately the parameters can be estimated, the better the predication of latency.

This is worthwhile when latency is important and questionable.

This is not worthwhile when it is obvious there is sufficient capacity to satisfy the demand.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Availability Modeling

innovate achieve lead

Another quality attribute with a well-understood analytic framework is availability.

Modeling an architecture for availability—or to put it more carefully, modeling an architecture to determine the availability of a system based on that architecture—is a matter of determining the failure rates and the recovery times of the components.

Just as for performance, to model an architecture for availability, we need an architecture to analyze.

Suppose we want to increase the availability of a system that uses the Broker pattern, by applying redundancy tactics.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

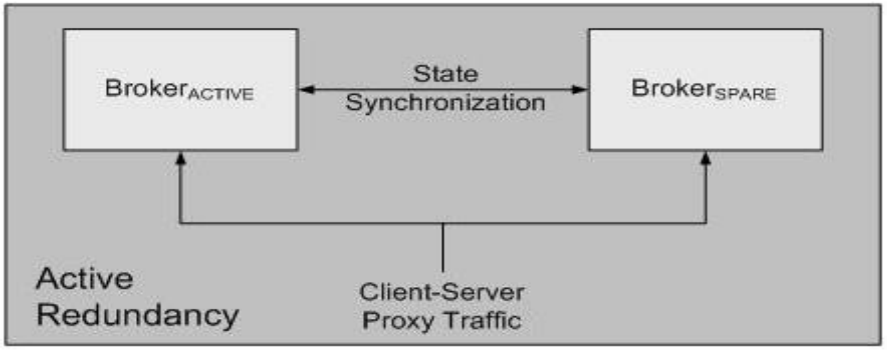
Availability Modeling

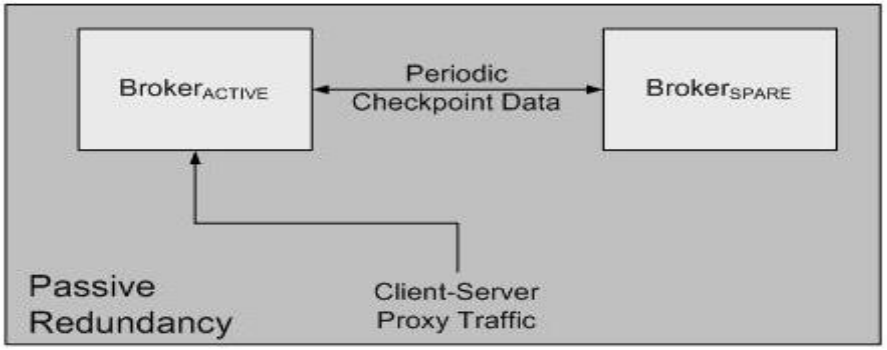
Three different tactics for increasing the availability of the broker are:

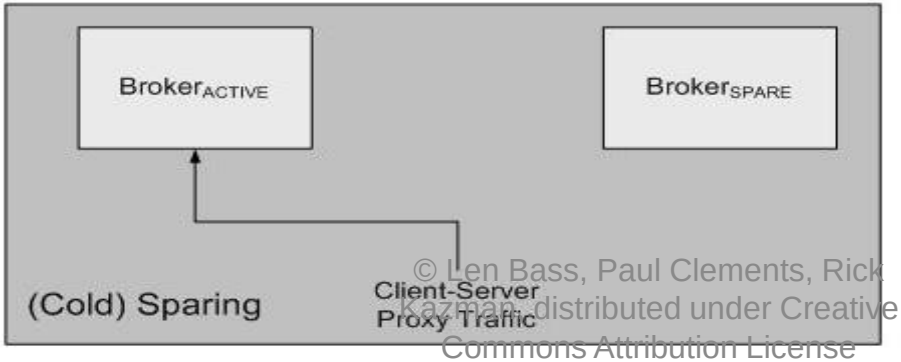
- active redundancy (hot spare)
- passive redundancy (warm spare)
- spare (cold spare).

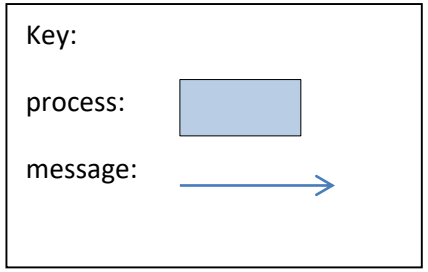
© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Making Broker More Available









© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

ed to be University under Section 3 of UGC Act, 1956

Applying Probabilities to Tactics

Using probabilities to model different tactics

- When two events A and B are independent, the probability that A or B will occur is the sum of the probability of each event: P(A or B) = P(A)+ P(B).
- When two events A and B are independent, the probability of both occurring is P(A and B) = P(A) • P(B).
- When two events A and B are dependent, the probability of both occurring is P(A and B) = P(A) • P(B|A), where the last term means “the probability of B occurring, given that A occurs.”

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Passive Redundancy

Assume

- failure of a component (primary or backup) is independent of the failure of its counterpart
- assume failure probability of both is the same: P(F)

Then probability that both will fail is: $1 - P(F)^2$

Can also estimate probability of failure given other tactics.

Then given a cost of implementing appropriate tactic we can do cost/benefit analysis

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Calculated Availability for an Availability-Enhanced Broker

Function	Failure Severity	MTBF (Hours)	MMTR (Seconds)		
			Active Redundancy (Hot Spare)	Passive Redundancy (Warm Spare)	Spare (Cold Spare)
Hardware	1	250,000	5	5	300
	2	50,000	30	30	30
Software	1	50,000	5	5	300
	2	10,000	30	30	30
Availability			0.9999998	0.999990	0.9994

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Maturity of Quality Attribute Models

Quality Attribute	Intellectual Basis	Maturity / Gaps
Availability	Markov models; Statistical models	Moderate maturity in the hardware reliability domain, less mature in the software domain. Requires models that speak to state recovery and for which failure percentages can be attributed to software.
Interoperability	Conceptual framework	Low maturity; models require substantial human interpretation and input.
Modifiability	Coupling and cohesion metrics; Cost models	Substantial research in academia; still requires more empirical support in real-world environments.
Performance	Queuing theory; Real time scheduling theory	High maturity; requires considerable education and training to use properly.
Security	No architectural models	
Testability	Component Interaction Metrics	Low maturity; little empirical validation.
Usability	No architectural models	

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Quality Attribute Checklists

A quality attribute checklist provides a means of:

- Checking requirements. Do the requirements capture all of the nuances of a particular quality attribute?
- Auditing. Does the design satisfy all of the aspects necessary for a certification process.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Security Checklists

innovate achieve lead

Security checklists are common.

- Vendors who accept credit cards should conform to the PCI (Personal Credit Information) standard
- Electricity producers have security checklists to prevent attacks on critical infrastructure

Checklists have both:

- *Product requirements*. E.g. the PCI checklist states that the security code on the back of the credit card should never be stored.
- *Process requirements*. E.g. patches should be applied promptly and there should be someone who has the organizational responsibility to ensure that they are.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Thought Experiments

innovate achieve lead

A thought experiment is mentally or verbally working through a particular scenario.

- Commonly done by the architect during design to explore alternatives.
- Also done during evaluation/documentation to convince third parties of the wisdom of particular design choices

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Thought Experiment Steps

Enumerate the steps of a use case

At each step, ask {yourself, the architect}

- What mechanism is being implemented to support the achievement of which particular quality requirement?
- Does this mechanism hinder the achievement of other quality attribute requirements?

Record problems for later deeper analysis or prototype building

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Back-of-the-Envelope Analysis

Analysis does not need to be precise or detailed.

Rough analysis serves for many purposes. E.g. “the volume of traffic generated by this source should be well within the bounds handled by modern infrastructure”

Only do deeper analysis for questionable areas or important requirements.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Experiments, Simulations, and Prototypes

Many tools can help perform experiments to determine behavior of a design

- Request generators can create synthetic loads to test scalability
- Monitors can perform non-intrusive resource usage detection.

These depend on having a partial or prototype implementation.

- Prototype alternatives for the most important decisions
- If possible, implement prototype in a fashion so that some of it can be re-used.
- Fault injection tools can induce faults to determine response of system under failure conditions.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Simulations

Event based simulators exist that can be used to simulate behavior of system under various loads

- Must create the simulation.
- Must have a variety of different loads and responses to check for.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Analysis During Requirements and Design

Different types of analysis are done at different stages of the life cycle

Requirements:

- Analytic models/back of the envelope analysis can help capacity planning
- Checklists can help ensure capture correct set of requirements

Design:

- Prototypes can help explore design options
- Analytic models or simulation can help understand potential bottlenecks
- Checklists can help determine if used a correct mechanism

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Analysis During Implementation or Fielding

Experiments and synthetic load tests can be used during the implementation process or after fielding

Monitors can be used after fielding to determine actual behavior and find bottlenecks.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Analysis at Different Stages of the Lifecycle

Life-Cycle Stage	Form of Analysis	Cost	Confidence
Requirements	Experience-based analogy	Low	Low-High
Requirements	Back-of-the-envelope	Low	Low-Medium
Architecture	Thought experiment	Low	Low-Medium
Architecture	Checklist	Low	Medium
Architecture	Analytic Model	Low-Medium	Medium
Architecture	Simulation	Medium	Medium
Architecture	Prototype	Medium	Medium-High
Implementation	Experiment	Medium-High	Medium-High
Fielded System	Instrumentation	Medium-High	High

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Summary

Analysis is always a cost/benefit activity

- Cost is measure of creating and executing the analysis models and tools
- Benefit depends on
  - Accuracy of analysis
  - Importance of what is being analyzed

Analysis can be done through

- Models for some attributes
- Measurement
- Thought experiments
- Simulations
- Prototypes

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License