Future-Proofing Your Systems

Rick Kazman

Rick.Kazman@gmail.com
Software Engineering Institute/CMU
and *University of Hawaii*

Motivating Quotations – 1

Business is a good game—lots of competition and a minimum of rules. You keep score with money.

-- (Atari founder) Nolan Bushnell

There's no business like show business.

-- Irving Berlin

There's no business like show business, but there are several businesses like accounting.

-- David Letterman

Motivating Quotations – 2.

I never get the accountants in before I start up a business. It's done on gut feeling.

-- Richard Branson

It has been my experience that competency in mathematics, both in numerical manipulations and in understanding its conceptual foundations, enhances a person's ability to handle the more ambiguous and qualitative relationships that dominate our day-to-day financial decision-making.

-- Alan Greenspan

Decision-Making in Architecture – 1

- Architectures are at the fulcrum of a set of business, social, and technical decisions.
- A poor decision in any dimension can be disastrous for an organization.
- A decision in any one dimension is influenced by the other dimensions.
- So the dimensions must *managed*, to be continually aligned.

Decision-Making in Architecture – 2

- Architecture decision-making is often done on an ad hoc, gut feeling basis.
- We, as an industry, are more like Richard Branson than Alan Greenspan.
- Why?

Decision-Making in Architecture – 3.

This talk =>

how to use economic decision-making to plan and manage architectures for evolution, so that alignment is optimally maintained over time.

The Goal of Economics-driven Architecting

- The application of *value-based* activities
  - that are practical
  - easily implemented
  - on a firm, principled basis
  - with simple, clear rationale
- We aim for Pareto-optimality (80-20 rules)
- What is an 80-20 rule?
- In 1906 Vilfredo Pareto noted that 80% of income in Italy went to 20% of the population.

Pareto Optimality

- Given a set of alternative allocations and a set of individuals, a movement from one allocation to another that can make at least one individual better off, without making any other individual worse off, is called a Pareto improvement.
- An allocation of resources is Pareto optimal when no further Pareto improvements can be made.
- How do we achieve this in practice?
- How do we apply Pareto optimality to value-based architecture decisions?

Value

- Value is how much a product or service is worth
  - Often measured in \$ or € or ¥, but does not have to be
    - Time, productivity, evolvability, reputation
    - Trickier to measure, but extremely useful nonetheless!
  - Value calculations may have many dimensions
    - includes schedule, personnel, material
    - includes what it is worth to the consumer, producer, maintainer…
  - Can sometimes be determined via market analysis:
    - e.g. how valuable is it for the software architect to have an evolvable design?

Value-driven View of Design

Realized in many ways, for example:

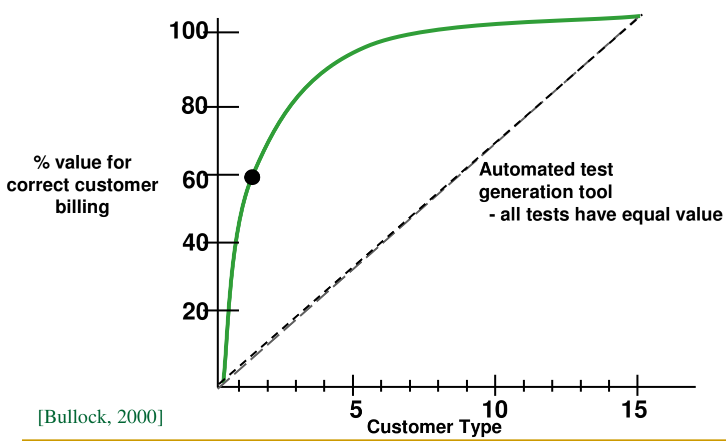
- Evaluating a given design
- Optimizing among design alternatives
- Exploring tradeoff spaces
- Exploring paths for evolution

Value Example: Software Testing

- Vendor proposition
  - Our test data generator will cut your test costs in half
  - We'll provide it to you for 30% of your test costs
  - After you run all your tests for 50% of your original costs, you're 20% ahead
- Any concerns with vendor proposition?
  - Test data generator is value-neutral*
  - Every test case, defect is equally important
  - But in reality, 20% of test cases cover 80% of business value

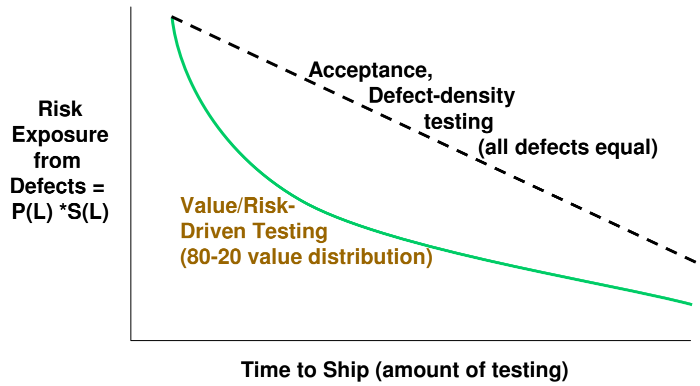
* As are most current software engineering techniques

20% of Features Provide 80% of Value: Focus Testing on These



[Bullock, 2000]

Resulting Reduction in Risk Exposure



Value in Software Engineering Today

- *Cost-driven* view of value prevails (e.g. Function Points, COCOMO)
- No value-based *design* heuristics tied to design decisions. Consider other disciplines:
  - Steel construction is more costly upfront, but is quicker to build => provides value via quicker time to market
- We lack techniques (and a common vocabulary) to talk about value in software
  - Can we talk about comparisons equivalent to “steel versus reinforced concrete” in software design?

Today's Talk

- Towards a method for explicitly optimizing *expected value* with respect to cost, schedule, constraints
- Architects are supposed to do this.
- But it's a multi-constraint search problem through a vast search space, with large amounts of uncertainty.

⇒ architects seldom do it well, or with confidence

⇒ even if they do it well, they can't communicate their rationale to decision-makers

Quantitative Decision Making

- Many valuation techniques used in IT are borrowed from financial economics:
  - Finance basics
    - Return on investment
    - Discounted cash flow
    - Net present value
  - Utility theory
  - Options theory

Quantitative Decision Making

- Many valuation techniques used in IT are borrowed from financial economics:
  - Finance basics
    - Return on investment
    - Discounted cash flow
    - Net present value
  - ***Utility theory*** ← our focus
  - Options theory

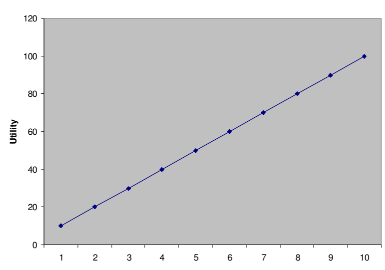
Using Utility to Guide Design

- Utility is a measure of the relative happiness or satisfaction gained from a good or service.
- Issues:
  - What level of utility can be achieved with your design?
  - What is at stake in designing for more utility?
  - How can you use this approach to elicit better information?
- Utility is useful to explore the trade-off space between requirements or within one requirement, e.g.
  - You want to know when more of something has no value, or when more of something with minimal effort buys you a lot

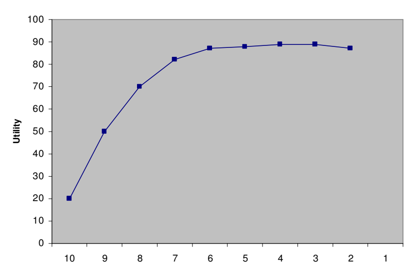
Quality Attributes

- Quality attribute requirements exert the strongest influence on architectural design.
- Quality attributes should be designed into the architecture
- Quality attribute requirements can be expressed in a common form.
  - Quality attribute scenarios with six parts: source, stimulus, artifact, environment, response, response measure
- Economics-driven trade-off analysis of quality attributes requires us to express quality attributes in a common form, prioritize them and use a common basis to compare them to each other.

Utility Curves and Quality Attributes

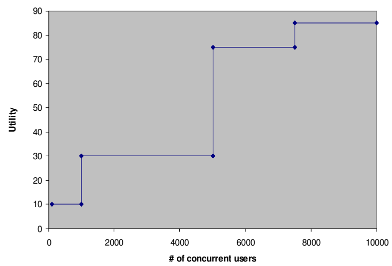


Example A: Utility to be gained increases monotonically as the resources improve

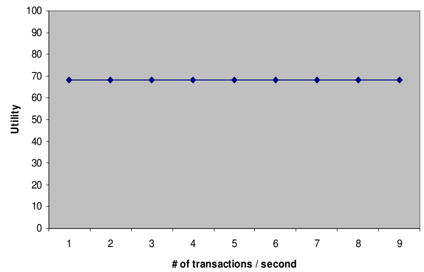


Example B: Achieving more of a quality attribute response does not always add more value

Utility Curves and Quality Attributes



Example C: Step function which may assist eliciting and designing for critical response measures where utility increases significantly



Example D: Constant function which may indicate that the stakeholders are indifferent to a particular quality attribute level

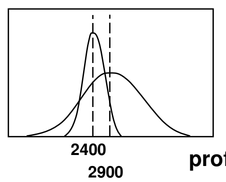
Investments

- In any investment you should consider the:
  - Potential benefit
  - Cost
  - Risk/uncertainty
- How do we quantify these when the investment is software?
- The CBAM (Cost Benefit Analysis Method) extends the ATAM framework to elicit and model costs, benefits, and uncertainty.

Example

	Design A	Design B
Avg Latency	500 ms.	200 ms.
Availability	99.9%	99%
Cost	3200	2600
Benefit	6100	5000
Profit	2900	2400

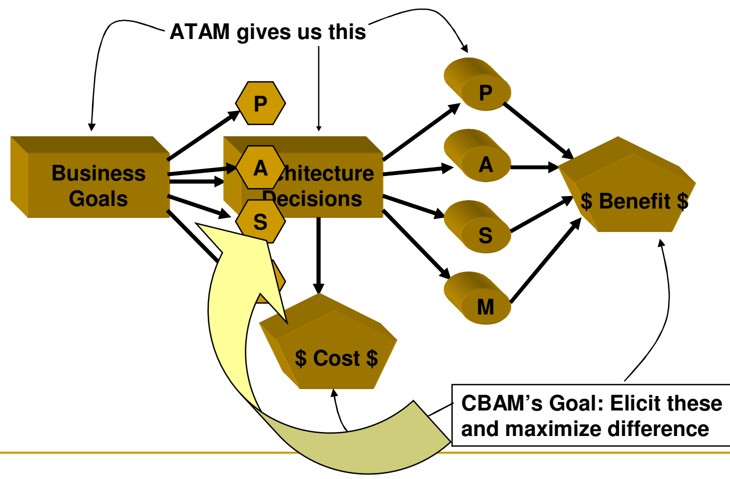
probability distribution



Ramifications of the Example

- This example is, of course, over-simplified.
- However, even this simple example brings up complex issues:
  - Which architectural decisions achieve these responses? What is their risk/uncertainty?
  - How risk averse are you?
  - How do you assess your level of uncertainty?
  - How do you compare the value of different system qualities?
  - What are the personnel/schedule implications of the architectural decisions?
- Which sources of uncertainty do you typically have to deal with and how would you go about characterizing, measuring, and minimizing them?

Context for the Work



The CBAM

- The aim of the CBAM (Cost Benefit Analysis Method) is:

to explicitly associate costs, benefits, and uncertainty with architectural decisions, as a means of optimizing the choice of such decisions.

- We assume that an organization has the ability to estimate costs:
  - COCOMO II, Function Points, SLIM, experience...
- The CBAM is focused on eliciting *benefits* and *uncertainty.*

Building Upon the ATAM

- When the CBAM commences, the following information must be documented:
  - The system's architecture-level design
  - The prioritized business goals of the system
  - The technical and business constraints
  - A ranking of the scenarios
  - The technical architectural decisions that are sources of uncertainty/risk in the existing architecture

The Steps of the CBAM

Starting from this base, we then execute the steps of the CBAM:

1. Collate scenarios.
2. Refine scenarios.
3. Prioritize scenarios.
4. Assign intra-scenario utility.
5. Develop architectural strategies (ASs) and determine quality-attribute-response levels.
6. Determine the utility of the expected quality-attribute-response levels by interpolation.
7. Calculate the total benefit obtained from an AS.
8. Choose ASs based on value for cost (VFC).
9. Confirm results with intuition.

The Iterations of the CBAM

- Typically making architectural decisions involves a significant amount of effort.
- To optimize the use of everyone's time, we split the CBAM into several iterations:
  - ***Triage*, where we quickly choose a set of architectural decisions to consider.**
  - ***Detailed Examination*, where we more carefully consider the costs, benefits, and interactions of a *subset* of the architectural decisions.**

An Example

- NASA's EOSDIS (Earth Observing System Data Information System) project, an enormous Web-based scientific information system:
  - **1.1 million lines of custom code**
  - **12,000 modules**
  - **50 COTS products**
  - **http://eospso.gsfc.nasa.gov/**
- The EOS is a constellation of satellites that gathers data about the earth for the U. S. Global Change Research Program.

1-3. Collate, Refine, and Prioritize Scenarios

- To make architecture investment decisions, we begin by asking what system scenarios are important for the business goals.
- Collate the QA scenarios elicited during the ATAM exercise.
- Prioritize based on satisfying the business goals of the system and choose the top 1/3 for further study.

1-3. Collate, Refine, and Prioritize Scenarios

- We collect scenarios from the stakeholders. Initially these are unrefined, e.g.

Scenario	Scenario Description
1	Reduce data distribution failures that result in hung distribution requests requiring manual intervention.
2	Reduce data distribution failures that result in lost distribution requests.
3	Reduce the number of orders that fail on the order submission process.
4	Reduce order failures that result in hung orders that require manual inter-vention.
5	Reduce order failures that result in lost orders.

1-3. Collate, Refine, and Prioritize Scenarios

- Refine the scenarios focusing on their stimulus/response goals.
- Elicit the worst, current, desired and best QA level for each scenario, e.g.

Scenario	Response Measure Goals			
	Worst	Current	Desired	Best
1	10% hung	5% hung	1% hung	0% hung
2	> 5% lost	<1% lost	0% lost	0% lost
3	10% fail	5% fail	1% fail	0% fail
4	10% hung	5% hung	1% hung	0% hung
5	10% lost	<1% lost	0% lost	0% lost

1-3. Collate, Refine, and Prioritize Scenarios

- Allocate 100 votes to each stakeholder and have them vote on the scenarios.
- Total the votes and choose the top 50% of the scenarios for further analysis.

Scenario	Votes	Response Measure Goals			
		Worst	Current	Desired	Best
1	10	10% hung	5% hung	1% hung	0% hung
2	15	> 5% lost	<1% lost	0% lost	0% lost
3	15	10% fail	5% fail	1% fail	0% fail
4	10	10% hung	5% hung	1% hung	0% hung
5	15	10% lost	<1% lost	0% lost	0% lost

4. Assign Intra-Scenario Utility

- How to compare the various scenarios? We need a shared measure of "goodness". We use *utility*.
- Determine the *utility* for each response level, e.g.

Scenario	Votes	Utility Scores			
		Worst	Current	Desired	Best
1	10	0	80	95	100
2	15	0	70	100	100
3	15	0	70	100	100
4	10	0	80	95	100
5	15	0	70	100	100

4. Assign Intra-Scenario Utility

- Note that in this step we are converting from technical measures (latency, mean time to failure, # of requests served per minute, etc.) to generic measures of *goodness*.
- This key step allows us to compare different quality attributes.

Scenario	Votes	Worst	Current	Desired	Best
3	15	10% fail	5% fail	1% fail	0%fail
		0	70	100	100

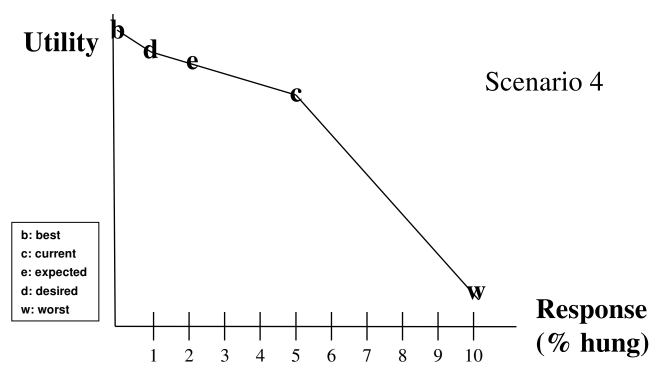
5-6. Develop ASs and Determine their QA Response Level

- Develop ASs that address the chosen scenarios.
- Determine the response levels that result from implementing these ASs.
- Call these the *expected* levels. We can interpolate their utility values.

AS	AS Name	Scenarios Affected	Current Response	Expected Response
1	Order persistence	3	5% fail	2% Fail
		5	<1% lost	0% lost
4	Order segmentation	4	5% hung	2% hung
5	Order re-assignment	1	5% hung	2% hung
6	Order retry	4	5% hung	3% hung

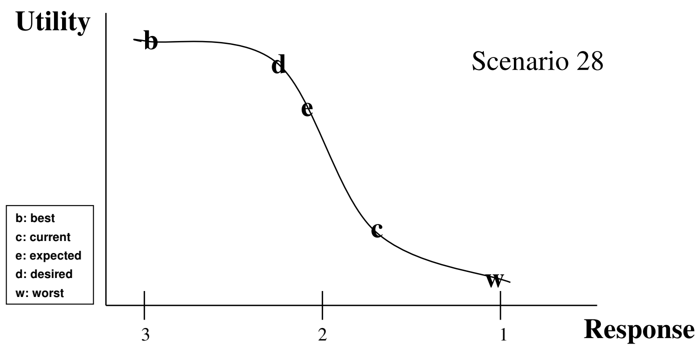
5-6. Develop ASs and Determine their QA Response Level

- What have we elicited and developed?
- A utility/response curve!



5-6. Develop ASs and Determine their QA Response Level

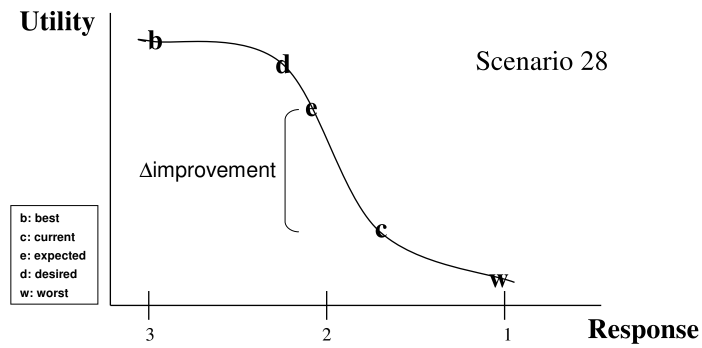
- These curves will be different for different scenarios.



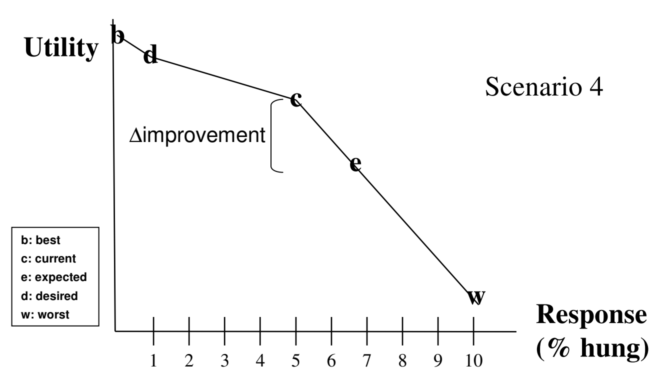
7. Determine an AS's Total Benefit

- Calculate the expected benefit of each architectural strategy ASi.
- For each scenario where ASi is used:
  - calculate the *Δimprovement* in utility as the difference between the 'current' level and the 'expected' level.
  - normalize this benefit amount using the votes collected in step 1
  - sum these normalized values

Positive ΔImprovement



Negative ΔImprovement



7. Determine an AS's Total Benefit

AS	Scenario	Benefit ΔUtility = Utility $_{expected}$ – Utility $_{current}$	Votes	Normalized Benefit (Benefit x Votes)	Total Benefit ΣScenario $_{Normalized\ Benefit}$
1	3	20	15	300	
	5	30	15	450	
	6	20	10	200	***950***
3	9	30	10	300	
	10	-5	5	-25	***275***
4	4	10	10	100	***100***
6	4	5	10	50	***50***

8. Choose ASs Based on "Value for Cost"

- Calculate the expected *cost* of implementing each architectural strategy AS_i that results in the expected benefit.
- Estimate the *schedule* implications of each AS_i in terms of person-months of effort and/or elapsed time.
  - **Note any contention for shared resources among these estimates (hardware, software, or personnel).**

8. Choose ASs Based on "Value for Cost"

- Now we can calculate the VFC (value for cost) ratio of each AS investment, and its rank.

AS	Cost	Total AS Benefit	AS VFC	AS Rank
1	1200	950	0.79	1
3	400	275	0.69	2
4	200	100	0.5	3
5	400	120	0.3	7
6	200	50	0.25	8
7	200	70	0.35	6

8. Choose ASs Based on VFC

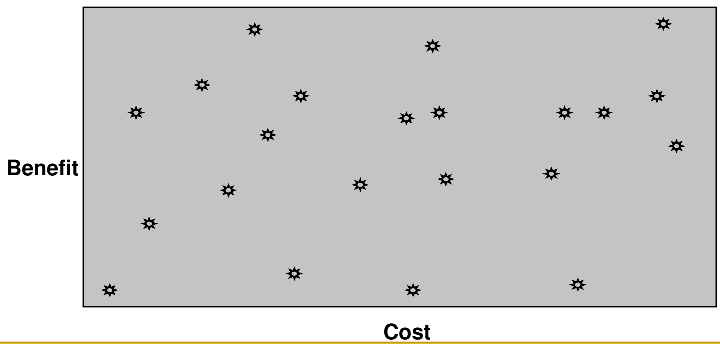
- We are now in a position to make *informed* decisions, based on:
  - the QA responses of each AS and their associated benefit,
  - the costs and schedule implications of implementing each AS

9. Confirm Results With Intuition

- Each CBAM step involves stakeholder input, and hence subjectivity.
- To ensure that the results are well-founded we examine the results, with respect to the *business goals* of the system.
- If the results conflict with intuition we need to determine if there are other issues that have not been considered while making these decisions.

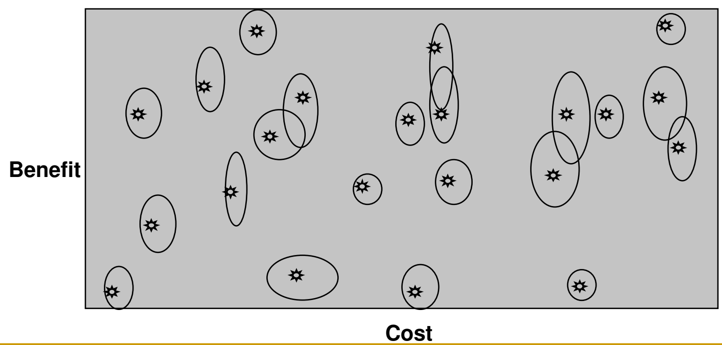
The Process (Recap)

- We select scenarios and ASs to address them.
- We elicit the benefits and costs of each AS.



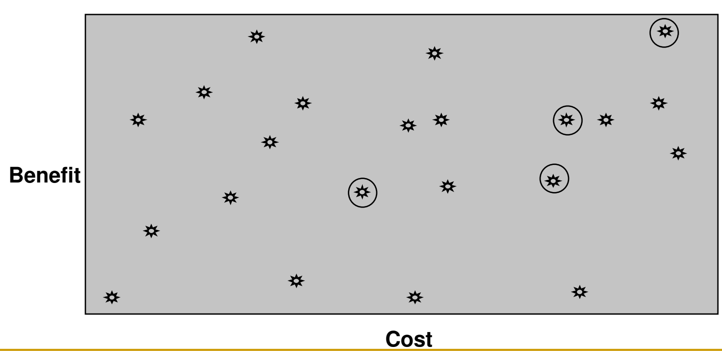
The Process (Recap)

- We can also determine the uncertainty associated with each AS.



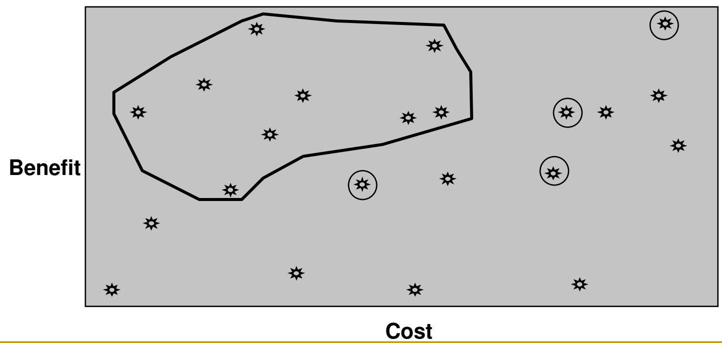
The Process (Recap)

- Some ASs *must* be chosen. Remove these from consideration.



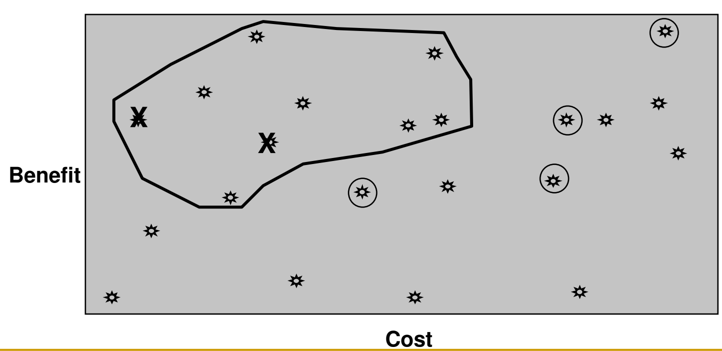
The Process (Recap)

- Now consider the set of high benefit, low cost ASs.



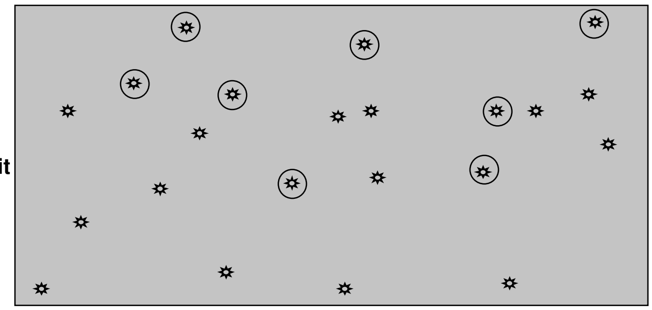
The Process (Recap)

- Some ASs may be excluded because of resource or time-to-market/schedule conflicts.



The Final Result

- Choose a final set to implement.
- Some ASs may be in/excluded because of dependencies.



The Result

- After this exercise, we have determined a set of ASs that address our highest priority scenarios.
- These chosen ASs represent the *optimal* set of architectural investments.
- They are optimal based upon considerations of:
  - **benefit**
  - **cost**
  - **schedule**
  - **uncertainty**
- This gives us a single step in evolving an architecture

An Economics-Based Method for Evolving Architectures

Final Motivating Quotation

It's tough to make predictions, especially about the future.

-- Yogi Berra

Supporting Evolution

- The CBAM addresses a single “snapshot” of architectural decisions.
- This is already complex.
- How do we think about evolution, which is a *trajectory* of architecture decisions?

Evolving An Architecture - 1

- Evolution is a process of strategic planning to maximize system value.
- Evolution starts with (and critically depends on) business strategy.
- But this strategy needs to be translated into an *actionable* plan for the enterprise architecture and, eventually, for individual system architectures.
- How do we do this?
- How do we maximize future system value in the face of uncertainty?

Evolving an Architecture - 2

- When trying to calculate future system value, you need to consider a series of events, motivated by business strategy, and represented by *strategic scenarios*.
- Maximizing system value is then an optimization problem => choosing an optimal set of

  [strategic scenario, architecture strategy]

  pairs
- How do we do this?

An Architecture Evolution Method - 1

ScenarioValue =
*Probability * Importance* *(1)*

Q: But what is *Importance*?

A: Importance can be calculated by estimating the size of the gain (or loss) associated with this scenario, along with its frequency.

An Architecture Evolution Method - 2

ScenarioValue =
*Probability * Size(Gain|Loss) * Frequency* (2)

Q: But how do we compare these values, given that the scenarios are not all exercised at the same time.

A: To properly compare them we need to consider the *Net Present Value* of the gain or loss.

An Architecture Evolution Method - 3

ScenarioValue =
*Probability * NPV(Size(Gain|Loss)) * Frequency* (3)

Q: How do we compute the *Size(Gain | Loss)* term?

A: Use ΔUtility and cost (as in the CBAM).

An Architecture Evolution Method - 4

$$ScenarioValue_{i,j} = Probability_i * NPV(\Delta Utility_{i,j} - Cost_j) * Frequency_i \quad (4)$$

We've arrived!

Choosing an evolution path involves choosing a set of AS_j such that:

$$\Sigma \text{ ScenarioValue}_{i,j}$$

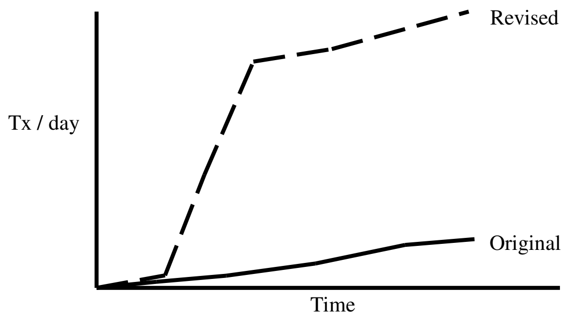
is maximized.

Reflections

- We started by claiming that evolution was a series of strategic scenarios.
- This is, of course, an over-simplification.
- In reality, we seldom consider series—a simple line—except in retrospect.
- For evolution we need to consider a multi-dimensional *space* of events, each of which has some conditional probability.

Reflections - 2

- Consider a single quality attribute: performance.



- How do we plan for uncertainty?

Architectural Support for Uncertainty

- Architectural strategies are real options
  - ⇒ provide architects the right, but not the obligation, to take subsequent design actions
- We cannot value architectural strategies by simply estimating their future utility along a small number of trajectories defined by a set of strategic scenarios
- We need to consider their value taking *uncertainty* into account.

An Architecture Evolution Method - 5

- Instead of using the term NPV(ΔUtility$_{i,j}$ - Cost$_j$) in formula (4), we will use a *real option valuation* (ROV) formulation of the architectural strategy
- Our formula now becomes:

$$ScenarioValue_{i,j} = Probability_i * ROV_{i,j} * Frequency_i \qquad (5)$$

Now we are in a position to describe a *method*.

A Method to Optimize Architecture-Based Evolution

1. Create a set of strategic scenarios
2. For each strategic scenario created rate:
   - *Probability of occurrence*
   - *Frequency of occurrence*
3. Assign intra-scenario utility
4. Develop architectural strategies (ASs) and determine QA response levels.
5. Determine the utility of each AS's expected QA response level by interpolation.
6. Calculate the total benefit obtained from an AS using the binomial options pricing model.
7. Optimize over all [*strategic scenario*, *AS*] pairs

Status

- Parts of this method are already road-tested.
- The method is currently being piloted in two engagements.
- Stay tuned!

Summary

- The CBAM is a method for optimizing architecture investment decisions, considering cost, benefit, and uncertainty.
- The evolution method helps stakeholders prioritize and cluster changes to an architecture.
- The evolution method builds upon the CBAM, but adds in more sophisticated modeling of benefit, dependencies, and uncertainty.

Further Resources

- L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, 2nd ed., Addison-Wesley, 2003.
- P. Clements, R. Kazman, M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley, 2001.
- R. Nord, *et al*, "Integrating the Architecture Tradeoff Analysis Method with the Cost Benefit Analysis Method", SEI Technical Report CMU/SEI-2003-TN-038, 2004.
- R. Kazman, J. Asundi, M. Klein, "Making Architecture Design Decisions: An Economic Approach", SEI Technical Report CMU/SEI-2002-TR-035, 2002.
- I. Ozkaya, R. Kazman, M. Klein, "Quality-Attribute Based Economic Valuation of Architectural Patterns", SEI Technical Report CMU/SEI-2006-TR-022, 2006.