Proceedings of the 5th International Workshop on Economics Driven Software Engineering Research (EDSER-5), affiliated with the International Conference in Software Engineering, May 2003, Portland, OR, USA

Time is Not Money

the case for multi-dimensional accounting in value-based software engineering

Vahe Poladian, Shawn Butler, Mary Shaw, David Garlan
{vahe.poladian, shawn.butler, mary.shaw, david.garlan}@cs.cmu.edu
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

ABSTRACT

"Time is money", or so goes the old saying. Perhaps influenced by this aphorism, some strategies for incorporating costs in the analysis of software design express all costs in currency units for reasons of simplicity and tractability. Indeed, in theoretical economics all costs can, in principle, be expressed in dollars. Software engineering problems, however, often present situations in which converting all costs to a common currency is problematical. In this paper we pinpoint some of these situations and the underlying causes of the problems, and we argue that it is often better to treat costs as a multidimensional value, with dimensions corresponding to distinct types of resources. We go on to highlight the differences among cost dimensions that need to be considered when developing cost-benefit analyses, and we suggest mechanisms for mediating among heterogeneous cost dimensions.

Keywords

Cost analysis, multi-dimensional cost analysis, value-based software engineering.

1. ACCOUNTING FOR COSTS IN SOFTWARE ENGINEERING

Although engineers traditionally focus on the functionality of their designs, they are becoming increasingly away of the need to address total cost of developing and owning the software.. A common approach to cost-benefit analysis is to express all costs and benefits in terms of dollars. To a first approximation, expressing all costs in a single dimension may seem like a reasonable solution. In practice, however, simplistic conversions of costs (or benefits) can be problematical.

Consider a military operations center, which is responsible for managing and directing battlefield assets during times of conflict. A typical operations center uses several applications that will demand different amounts of computer resources (e.g., bandwidth and CPU resources) depending on the military situation. For example, satellite and air reconnaissance assets can provide real-time video coverage of the operational area, but not all the time. Simultaneously, communication channels stream important intelligence and operational information to the center's military commander, but processing the messages is CPU intensive. Unfortunately the amount of information available to the commander can exceed his capacity to receive and process the information and affect his ability to make informed decisions.

The value of each application, and thus the value of the computing resources, will depend on the current military situation. For example, at times the commander will need very detailed videos of the battlefield to make operational decisions, but at other times the commander will need to receive intelligence over communication channels and a less detailed picture of the battle will be adequate. Therefore, dynamic reconfiguration of the computing resources may be essential to making timely military decisions.

The quality of service that these applications provide can be fine-tuned through computer resource adjustments to meet the commander's needs. For example, increasing frame rates and bandwidth allocations can enhance video imagery, but the resulting demand for CPU cycles to process video images can cause delays in message processing.

At any given point in time, finding the optimal allocation of computing resources depends on the value that each application provides to the commander. Finding the optimal resource allocation ultimately requires all the alternatives to be comparable -- typically expressed in a common metric. However, there can be serious drawbacks to making these conversions too early in the analysis process.

First, it is difficult to associate cost or value with a resource without complete information about the resource and the context of its use. In the example above, the value of bandwidth was highly dependent on the battlefield situation and weather. In some cases the value of the resource may be in saving lives, but in other situations the value of the resource may be tied to common economic costs, such as fuel or energy costs, which are more easily translated into dollars[1].

Second, a conversion between metrics may adversely affect the type of analysis that you can do. The value of the resource may be non-linear with respect to the preferences of the user (or commander in the example above). Converting the value of computing resources, such as bandwidth and CPU, to dollars implies that each of the resources is as finely divisible as dollars. Increasing the resolution of video images requires a stepwise increase in bandwidth before the user recognizes a difference in the quality of the video. As a result, calculus-based solutions may appear adequate to solve the problem in the abstract, when in fact discrete algorithms are more appropriate for the problem at hand.

Third, conversions to a common currency can lose information that should affect the types of feasible solutions. Some resources are perishable: they are only valuable for short periods of time, and after that they have no residual value. Converting such a resource to one that is not perishable will cause important information to get lost in the process. In fact, the obtained result may not be feasible according to the original formulation of the

[1] Despite studies that calculate the price of an individual's life, few decision makers are willing to make explicit comparisons.

problem. For example, unused bandwidth is gone forever, and allocating bandwidth to satellite imagery when the satellites are not overhead is not a feasible solution.

These problems can be avoided by using methods that recognize and respect the different properties of different resources. Here we regard cost as a multidimensional quantity, with different dimensions corresponding to different non-commensurable cost metrics. Section 4 presents examples based on two such methods. One of the examples focuses on the automatic run-time configuration of software components based on preferences of the user. The second example tackles the problem of choosing the optimal set of countermeasures to minimize threats to a corporate IT infrastructure.

In this paper, we contribute to understanding the problem on incommensurable multidimensional costs and finding solutions for particular projects by:

- *Characterizing types of costs to show their differences.* In Section 2 we catalog resources that are commonly used in analyzing costs in software development.
- *Proposing a model for treating cost as a multidimensional measure.* In Section 3 we present a model that explains the essential differences among these costs.
- *Analyzing the problems of mapping among cost dimensions.* In Section 4 we discuss analysis techniques that carry through multidimensional costs.
- *Showing how to accommodate methods that require uni-dimensional costs.* In Section 5 we generalize from the examples of Section 4 and discuss ways to balance the information needs that require multiple dimensions with the analysis needs that require a single dimension.

2. SOURCES OF SOFTWARE DEVELOPMENT AND QUALITY COSTS

In software engineering and systems research, cost-benefit analyses have been used to solve cost estimation and optimization problems. SAEM [2], [3] is a cost benefit analysis model that helps security managers choose the best set of countermeasures. Odyssey [9] provides runtime middleware that helps adapt application behavior to resource availability. The Nemesis [10] operating system uses shadow prices and careful accounting to determine optimal allocation of resources among competing applications. Aura [8] aims to reduce user distraction in interactive computing by accounting for human attention as a resource. COCOMO II [1] is a software cost estimation model that calculates the cost of a software project based on various organizational and project parameters. The works cited consider costs and benefits to estimate benefit, determine optimal allocations, and estimate cost.

Table 1 catalogues some of the resources that are considered by the analyses of the research described above. This list is representative rather than complete; it provides the basis for the examples we use in later sections.

Table 1. A Selection of Resources and their sources

Cost dimension	*Examples, citations*
Purchase cost, currency (dollars, for simplicity)	Classical Economics
Staff time	COCOMO [1]
Reputation	SAEM [2]
Lives lost	SAEM
Calendar time, days	COCOMO
Bandwidth	Odyssey [9], Nemesis [10]
Battery Capacity Remaining	Odyssey, Nemesis
User attention	Aura [8]
Software application, e.g. Microsoft Word	Aura

3. PROPERTIES of COSTS/RESOURCES

The introductory example motivates the need to consider separate resource dimensions in cost-benefit analysis. But what are some of the characteristics of different resources that need to be considered during such analyses? Further, how would these characteristics influence the choice of analysis technique? To help answer these questions, we discuss the different properties of some of the resources identified in Table 1, with particular focus on understanding how these differences can be reconciled or mediated. At the end of the discussion, we summarize our findings in Table 2 for a sample of resources.

- *Divisibility/granularity.* This property describes how dense the space of the resource is. Intuitively, this property indicates in what increments the resource can be allocated. Possible values are:
  - *Continuous*: The resource can be allocated at a very fine grain. Bandwidth and battery energy are resources that fit in this group.
  - *Discrete but dense*: The possible allocation points are many, but allocation can not be made continuously. Currency fits this group.
  - *Sparse discrete:* There are very few possible points in the resource space. Editing a document with a particular application, e.g. Microsoft Word, falls in this category.

  The granularity of a resource can influence the choice of the solution method. With continuous resources and in some cases discrete dense resources, calculus-based solutions work well, especially if resource requirements can be described as closed formulas. Sparse discrete resources are best analyzed with discrete methods such as integer programming and knapsack algorithms. Problems with continuous and dense discrete resources can also be tackled using discrete solutions, at the expense an approximate answer. This can be a justified trade-off if no closed-form formulas exist to describe the functions.
- *Fungibility.* This property describes whether a particular resource can be converted to another resource. This property makes sense in the context of a specific problem, and with respect to specific other resources. For example,

- *Complete fungibility:* Common currency is fungible to most other resource.
- *Partial fungibility:* Some interchange is possible between bandwidth and CPU cycles in the software runtime configuration problem. Consider different MPEG decoders using different compression algorithms. One decoder may be relatively bandwidth intensive, while the other may be CPU intensive. Availability of multiple decoders makes it possible to convert between bandwidth and CPU cycles. It is important to realize that the tradeoff is limited a few points.
- *No fungibility:* In software cost estimation problem, it is well known that calendar days and staff months are not interchangeable. Additional staff may even lengthen development time.

- *Measurement Scale.* This property describes the kind of scale that is appropriate for measuring a resource. For example, the set of domestic animals (dog, cat, cow, etc) has nominal scale, as there is no ordering relationship between elements in that set. See the Appendix for a review of measurement scales. Possible choices are:
  - *Nominal.*
  - *Ordinal.*
  - *Integer.*
  - *Ratio.*

  Cost-benefit analysis can sometimes be tackled by converting all resources to the same scale. However, conversion among resources of different scales must be made only when conversions are justified. See Section 4.2 for an example of such conversion.

- *Economies of scale.* This property describes the extent to which a percentage increment in a resource affects the increment of the output of a product that uses the resource as input. Possible values are:
  - *Superlinear Scale:* (also known as positive economies of scale). If a percentage increment in a resource results in proportionately higher increase in output, then we say resource has superlinear scale. Consider the problem of searching for a given record by its unique key in a large database. As a measure of output, consider the size of the database (e.g., total number of records) we are able to search in a fixed amount of time, and as a measure of input, consider the size of hardware we need to have (e.g., CPU speed). Recall that the binary search algorithm runs in time logarithmic with respect to number of items. Thus, CPU size exhibits superlinear scale with respect to the problem size in this case, because incremental increases in the CPU performance dedicated to the search space result in increasingly proportionally larger search space covered.
  - *Linear Scale:* (also known as neutral economies of scale). The benefit of additional quantities of the resource is independent of the problem size.

Table 2. Properties of Sample Resource Dimensions

		Properties of Costs						
		Units	**Measurement Scale**	**Granularity**	**Fungibility**	**Perishability**	**Economies of Scale**	**Rival**
Cost Dimension	Purchase cost	Dollars	Ratio	Dense	Y	N	Linear	Y
	Staff time	Months	Ratio	Sparse	N	Y	Sublinear	Y
	Reputation	Scale	Ordinal	Sparse	N	N	N/A	Y
	Lives lost	Number of Humans	Integer	Sparse	N	N/A	N/A	N/A
	Calendar time	Days	Ratio	Sparse	N	Y	Depends	N
	Bandwidth	Mbps	Ratio	Continuous	N	Y	Depends	Y
	Battery	Joules	Ratio	Continuous	N	N	Depends	Y
	Human attention	Seconds	Ordinal	Sparse	N	N	N/A	Y
	Software application	N/A	Nominal	Sparse	Y	N	N/A	N

- *Sublinear Scale:* (also known as diseconomies of scale). If a percentage increase in resource results in proportionately smaller increase in output, then we say the resource exhibits sublinear scale. Staff size, as input to software projects, exhibits slight diseconomies of scale (COCOMO II).

Notice that this property only applies to resources that are measured on a ratio scale or that can safely be converted to ratio scale.

- *Perishability.* This property describes whether the resource will be forever lost, if not used by certain point in time. Possible values are perishable and non-perishable.
  - *Perishable:* Bandwidth is perishable.
  - *Non-perishable*: Battery energy is not perishable.

Problems involving perishable and non-perishable resources need to introduce time into the analysis and account for intertemporal possibilities. Utility functions are one possible solution.

- *Rival.* A rival resource is such that the consumption of a unit or amount of a resource by one person or entity precludes the consumption of the same unit by another person.
  - *Rival:* Money, labor, bandwidth, CPU cycles.
  - *Non-rival:* Software application, information goods, calendar days.

Efficiently allocating rival resources among multiple requestors is the heart of many optimization problems. Aggregate demand for a rival resource can not exceed total supply available. Allocation analysis can be complicated when multiple sources of a resource are available. For example, consider the problem of choosing where to run a particular software application, given a choice of two servers.

4. MULTIDIMENSIONAL ANALYSIS TECHNIQUES

In this section, we present two examples of techniques that consider multiple dimensions of costs in solving cost-benefit problems in practical software systems. The first analysis is performed at run time and helps configure software applications on a mobile computer. The second analysis is performed off-line and optimizes the selection of security technologies to counter threats against corporate IT infrastructure.

4.1 Value-based Software Runtime Configuration

Let's revisit the scenario from the introduction and illustrate some of the problems that can result from early conversions. Tables 3 and 4 show hypothetical runtime operational profiles of the two programs described in Section 1: Messaging and Real-time Video. The quality level information in the first column is provided by the application specification. The second and third columns give resource usage (percentage-of-resource-required/second) to achieve the specified quality level. The resource data depends on the runtime characteristics of the application and the data processed, which can be obtained using profiler tools. The value information in the fourth column is represents the value assessments of the battlefield commander, which can be obtained through elicitation interviews.

The overall objective is to maximize the sum of the values: Value(Messaging) + Value(real-time video). Notice that the quality level is an ordinal scale: it does not make sense to say how much more or by what factor the next level is better than the previous one. A value function, which normalizes the commander's value assessments, converts the quality levels into a ratio scale on the basis of additional information elicited about the application. Bandwidth and CPU are both perishable resources and cannot be stored for future use.

Tables 3. The Operational Profiles of the Applications

Messaging

Quality Level	CPU, %	BW, %	Value
None	0	0	-∞
Very Low	57	17	1
Low	61	23	12
Medium	72	27	55
High	79	29	68
Very High	98	32	75

Real-time Video

Quality Level	CPU, %	BW, %	Value
None	0	0	-∞
Bad	12	43	3
Acceptable	19	52	30
Good	23	69	45
Very Good	27	78	57
Excellent	34	93	89

One approach to solving this problem is to take as given the external prices of CPU and Bandwidth, and convert these to a common currency. Assume the cost of one percent of available CPU is 2 units, and that of one percent of the available Bandwidth is 3 units. A total of 2 * 100 + 3 * 100 = 500 units of total resource are available. Table 4 present the resource requirements in terms of the single currency. Column 2 shows the cost in common currency of providing that level of quality, and column 3 shows the percentage of that cost.

Tables 4. The Operational Profiles Using Common Currency, 500 Units Available

Messaging

Quality Level	Cost	Cost, %
None	0	0
Very Low	165	0.33
Low	191	0.38
Medium	225	0.45
High	245	0.49
Very High	292	0.58

Real-time Video

Quality Level	Cost	Cost, %
None	0	0
Bad	153	0.31
Acceptable	194	0.39
Good	253	0.51
Very Good	288	0.58
Excellent	347	0.69

Notice that according to Table 2, the best combination that can be achieved is High quality of Messaging and Good quality of Real-time Video, which costs 498 units, or just under 100%, and is valued at 113. However, after consulting Table 3, we notice that CPU would be utilized at 101 percent, making that combination unattainable. The problem is that we have allowed conversion of unused Bandwidth into CPU, despite the inappropriateness of this conversion. Indeed, each quality point for either application can be obtained using only a unique CPU and bandwidth vector. The root of the problem is that CPU and bandwidth are not fungible, and our assumption of fungibility leads to an incorrect solution.

Another approach to this problem is to use derivatives, e.g. a calculus method called Lagrange Multipliers. However, since the space of quality points is sparse, any kind of continuous approximation is likely to yield a solution that is also not in the space of available quality points.

Currently, we are investigating the use of a Multidimensional, Multiple-Choice 0-1 Knapsack algorithm for handling this type of problem [11]. The solution to that problem is similar to the uni-dimensional version, except that it uses a parameterized vector for resource prices, and it iteratively refines the value of the parameters to eventually determine accurate conversion prices.

This technique can be extended to handle perishable resources, such as battery energy. In this case, intertemporal choices must also be considered, and an explicit function must be introduced to measure the value of saving energy for future use.

4.2 Security Attribute Evaluation Method

Traditional security risk management techniques advocate that security managers determine an organization's risk of an attack (*a*) by calculating the probable cost of the attack, i.e. $risk_a = cost * p(a)$, where *p* is the probability of the attack. For example if a virus attack results in *x* hours of lost productivity, then the risk of the attack is typically determined as $\mathrm{Risk}_{\mathrm{virus}} = x *$ average hourly wage rate * p(virus). Converting lost productivity to dollars appears relatively straightforward, but other types of attack consequences such as damaged public reputation or impaired quality of patient care are not as easily converted to dollars.

Unfortunately, simplistic risk calculations such as the one just described do not capture the value that organizations place on different types of costs. First, security managers find it difficult to attach explicit financial value to intangibles, such as public reputation or quality of patient care. Second, even when explicit economic value can be assessed, business executives are often skeptical about the underlying assumptions and lack confidence in the results. For example, organizations are usually less concerned about lost productivity from an attack than direct financial loss. Therefore, techniques that preserve the value of the outcome may produce more convincing results.

The Security Attribute Evaluation Method (SAEM) [2] uses multi-attribute decision analysis techniques to help security managers choose the best set of countermeasures against possible attacks. Although the SAEM risk assessment process reduces costs to a common *threat index*, the organization's value of each type of cost is captured as part of the threat index.

The risk assessment cost dimensions are the *most-likely* types of consequences of a successful attack, e.g., revenue lost, staff hours lost, reputation damage suffered. Security managers determine these cost dimensions. In order to determine the best set of counter-measures, SAEM calculates the relative importance of each consequence. The method introduces value functions to assess the incremental importance and normalize consequences, and uses the SWING-weight method [5] to elicit the importance of each consequence. Finally, SAEM computes the threat index, which is essentially a common, but neutral, cost measure that indicates the relative costs of an attack to other attacks.

5. RECONCILING MULTIDIMENSIONAL ANALYSIS WITH ONE-DIMENSIONAL TECHNIQUES

We have argued that cost-benefit analyses often need to maintain multidimensional representations of costs in order to preserve information about qualitative differences among distinct types of costs. We identified some of the principal characteristics of costs that impede conversion to common units, and we showed the consequences of failing to preserve the distinguishing information.

Eventually, though, we need to make decisions. To do so, we must be able to compare multidimensional costs. Further, some analysis techniques require scalar costs; the value of the analysis, even with loss of information, may be large enough to offset the information loss.

We believe that an appropriate strategy is to preserve the distinctions among different costs as long as practical and to reduce the cost vector to a scalar when circumstances force the conversion.

Consider, then that a system with N cost dimensions is being evaluated in an N-dimensional space, and assume for simplicity that the dimensions are orthogonal. Then each cost point is described by its cost in all the dimensions and corresponds to a point in space. The vector from the origin to that point represents that cost, and the length of that vector is one-dimensional. The problem is, how can we establish a value for the length of the vector? It is clearly inappropriate to treat the indices for the various dimensions as if their units were equivalent. Instead, we believe the proper approach is to preserve the N-dimensional analysis as long as possible, then perform late binding on the conversion by assigning a conversion function from each dimension into some common units. This makes it possible to compute the vector length and reduce the cost vector to a scalar. Vectors in each dimension can be scaled using parameterized weights, and then a common cost can be computed using root-mean-square as if it were Cartesian. This process can be iterated several times in order to achieve more accurate weights. Other approaches may be possible as well.

6. ACKNOWLEDGMENTS

This research is supported by the National Science Foundation under Grants ITR-0086003 and CCR-0113810, by the Sloan Software Industry Center at Carnegie Mellon, by DARPA under contract F30602-00-2-0616, and by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298. Authors would like to thank the members of the CMU Software Research Seminar for their critical feedback.

7. REFERENCES

[1] Barry Boehm. *Software Cost Estimation with COCOMO II.* Prentice Hall PTR, New Jersey: 2000.

[2] Shawn A. Butler. Security Attribute Evaluation Method. A Cost-Benefit Approach. *Proc ICSE 2002 - Int'l Conf on Software Engineering*, 2002

[3] Shawn A. Butler and Paul Fishbeck. Multi-Attribute Risk Assessment. *Symposium on Requirements Engineering for Information Security*, 2002.

[4] Shawn A. Butler and Mary Shaw. Incorporating Nontechnical Attributes in Multi-Attribute Analysis for Security. *Proc EDSER-4: Workshop on Economics-Driven Software Engineering Research*, 2002.

[5] *Proceedings of the Workshops on Economics-Driven Software Engineering Research*, EDSER-1, -2, -3, and -4. Workshops held in conjunction with the 21st through 24th ICSE's: International Conference on Software Engineering, 1999 to 2002.

[6] L. Briand, K. El-Emam, and S. Morasca. On the Application of Measurement Theory in Software Engineering. *Empirical Software Engineering*. 1(1), 1996.

[7] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous & Practical Approach*, International Thomson Computer Press, 1997.

[8] D. Garlan, D.P. Siewiorek, A. Smailagic, P. Steenkiste.. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing* 1(2), April-June, 2002.

[9] J. Flinn, M. Satyanarayanan. Energy-aware Adaptation for Mobile Applications. *Proc 17th SOSP - ACM Symposium on Operating Systems and Principles,* 1999.

[10] R. Neugebauer and D. McAuley. Congestion Prices as Feedback Signals: An Approach to QoS Management. *Proc 9th ACM SIGOPS European Workshop*, 2000.

[11] Vahe Poladian, David Garlan, and Mary Shaw. Software Selection and Configuration in Mobile Environments: A Utility-Based Approach. *Proc EDSER-4 - Workshop on Economics-Driven Software Engineering Research, 2002*.

Appendix: Quick Review of Measurement Theory

Not all measurements are created equal. More precise initial measurements enable more precise analyses and conclusions. Measure theory provides models that explain the differences and limitations.

Most members of this community are already familiar with this material, but many have forgotten the terminology. As a reminder, measure theory recognizes a number of scales for classification or measurement, ordered from less to more powerful [[6],[7]] The table summarizes the characteristics of the major scales.

Some examples of ways these scales can be abused help to show how the character of our data constrains the way we should use it:

"The temperature in Miami is 20 degrees Celsius, the temperature in Pittsburgh is 10 degrees, so it's twice as hot in Miami." *Wrong*. Celsius is an interval scale, and this kind of comparison is only valid in ratio or absolute scales. The Kelvin temperature scale is a ratio scale, so it's ok to convert to Kelvin and compare: "The temperature in Miami is 293 degrees Kelvin, the temperature in Pittsburgh is 283 degrees Kelvin, so it's 7% warmer in Miami."

"We surveyed the population for preferences on a scale of Strong Yes / Yes / OK / No / Strong No and coded the results on a 5-point scale with Strong Yes as 5 and Strong No as 1. Option A averaged 4.0, option B averaged 3.0, and option C averaged 2.0. Therefore option A dominated option B by as much as option B dominated option C." *Wrong*. The preferences are measured on an ordinal scale, and the comparison requires at least an interval scale. This sort of comparison is especially noxious when coupled with comparisons of the costs of the options. This is the kind of problem we're addressing in this paper.

Scale	*Intuition*	*Preserves*	*Example*	*Legitimate transformations*
Nominal	Simple classification, no order	Differences	Horse, dog, cat	Any one-to-one remapping
Ordinal	Ranking according to criterion	Order	Tiny, small, medium, big, huge	Any monotonic increasing remapping
Interval	Differences are meaningful	Size of difference	Temperature in Celsius or Fahrenheit	Linear remappings with offset (ax+b)
Ratio	Has a zero point	Ratios of values are meaningful	Absolute temperature (Kelvin), values in currency units	Linear remappings without offset (ax)