Paper # 5175

PRETTY SOON IT'S REAL MONEY

Jim Cook

Data Strategies

Systems Management Consultants

SOFTWARE DESIGN & INFORMATION TECHNOLOGY MANAGEMENT

9 Aberdeen St. Irvine, CA 92720-3302

(714) 552-4482

Abstract

The late senator Everett M. Dirksen, speaking about government spending, once remarked, "A billion here, a billion there and pretty soon, its real money." Information technology providers have spend billions over the years, ostensibly to provide economic value to their organizations, but they continue to manage these expenditures technically, rather than as a financial investment. It is no wonder then, that technology continues to proliferate while the returns it provides continue to fall short of the investors' expectations.

If we analyze the traditional technology management disciplines: availability management, performance management, capacity management, etc., it becomes obvious that their raison d'être is to improve the economic return on the technology investment. For example, the expense of providing system availability can only be justified if the losses incurred when the system is unavailable substantially exceed the cost of making it so. Similarly, the cost of performance enhancement is a direct trade-off against the cost of additional capacity which would provide the same performance. Another way to think about it is, if information technology were absolutely free, there would be little need to manage it.

Given that technology management is essentially a financial endeavor, involving exchanging a lesser, quantifiable expense for a greater, often not quantifiable one, it seems reasonable that the key indicators of technology management should be financial ones. How much was spent? By whom and for what? What is the cost of a unit of technology? How has that cost changed over time? How does the economic return on technology investments compare to that of non-technology investments? What is the best economic choice between alternative technologies? Answers to these questions would remove much of the uncertainty surrounding technology management and focus its deployment on the core mission of supporting the business.

Unfortunately, the financial management of technology isn't easy and the traditional tools and methodologies have not done much to make it easier. The current state-of-the-art, chargeback to end-users for technology-specific metrics (e.g. CPU time and disk storage) do more to obscure than highlight IT's contribution to the business. In fact, it is highly likely that tactical decisions made on the basis of traditional chargeback data will be wrong strategically. What is required is a completely different approach, which utilizes business-based rather than technical metrics, within a methodology of sound financial management practice.

This paper will outline how such metrics can be obtained using Hewlett-Packard's Application Response Measurement (ARM) technology. It will also describe in detail how these metrics can be utilized within a practical, activity-based costing methodology using a new approach developed by the author called, Delivery-Level Architecture.

Why Manage Technology?

Information technology management is frequently assumed to be a technical function, performed by technologists, to satisfy technical requirements. Those assumptions engender management practices which tend to optimize technological considerations by design and business considerations by accident, if at all. In reality, technology is a business enabler and its management should be focused on optimizing the business process or segment which is being enabled.

In this business-centric view of technology, the only significant considerations are cost of deployment and return on investment. Technical considerations like capacity, performance, and even service level are merely a means to express some aspect of cost-effectiveness.

The truly fundamental nature of financial performance in technical decision-making can be easily discerned through a simple thought experiment. Imagine that information technology, in all its forms, was completely free. Processing, storage, communications bandwidth, all could be had in unlimited quantity, simply for the asking. In this engineer's utopia, what technology management disciplines would be required?

Obviously capacity planning would be unknown, since it would be simpler just to acquire enough capacity to ensure an excess. Similarly, performance management would be unnecessary because even poorly designed and resource intensive applications could be made to perform adequately with sufficient capacity. Even service-level management would be unnecessary because nothing would prevent configuring the environment to ensure maximum service to all applications and users. And outages would become a thing of the past since all components could be made redundant. In fact, about the only management discipline that would be necessary is acquisition management to cope with the avalanche of free hardware and software.

Obviously, technology management is financial management, because, without the financial incentive, it simply disappears. The goal of any effective technology management program must then be to maximize its impact on its primary, financial objective.

NBUs, SLAs and the Quest for the Holy Grail

Management strategies which are based on measurable objectives tend to achieve (or at least move towards) those objectives over time. However, if the objectives do not measure the actual desired results, attaining them will not ensure success. It is no wonder, therefore, that technical management objectives do not ensure positive business results. The logical solution is to select business drivers as information technology objectives.

Over twenty years ago, Mario Morino, founder of Morino Associates (which begot Legent Corporation, which was subsequently acquired by Computer Associates), published a paper suggesting that technology management be based on business transactions rather than technology-dependent metrics. Over time, the mainframe community began to refer to such metrics as *Natural Business Units* or NBUs.

The problems associated with implementing technology management based on NBU's were well described in Morino's paper and haven't changed significantly since. They are:

- Lack of a standardized source of NBU metrics
- Requirement to modify application code to capture NBU's

Lack of Standards

Unlike the more commonly used resource metrics (*e.g.* CPU time), NBU's are typically defined to be an event within a particular business process. Because these processes tend to be enterprise-specific, NBU definitions are seldom "portable". This lack of commonality prevented vendors from supplying turnkey metrics solutions and required users to invest heavily in metrics definition.

Application Modification

Even more significant was the requirement to modify application software to capture NBU's. Typically, this involved locating the end-point of a business transaction within the application's instructions and inserting a subroutine call to a metrics logging routine. While not directly difficult, this did require application's programmers to analyze their code and make what they perceived to be "unnecessary" changes. In many cases, these programmers generously estimated the time required to make the changes (unlike the way they tended to estimate every other programming task), with the result that the costs became more than the enterprise was willing to bear.

Not surprisingly, in the face of these two obstacles, the use of NBU's was extremely limited. This perceived lack of interest translated into vendor apathy for NBU support which evolved into a self-fulfilling prophecy. I personally recommended, in my role as Senior Consultant to Morino Associates' Financial Products Group, that the company develop and market a business transaction metric collection facility. I initially made this recommendation in 1986 and subsequently about every other year until 1992. Each time the proposal was rejected because of fears that it could not be made functionally rich enough to attract a reasonable market share.

Distributed Systems Accounting and NBUs

In the mid-eighties, as a consultant to Legent Corporation, I began to look closely at the issues involved in accounting for distributed systems. It was immediately obvious to me that this environment presented an entirely different problem from the mainframe, specifically because multiple platforms were involved[1]. As I saw it (and continue to believe), the key was not to be able to charge for the work done on a specific platform, but to be able to charge the same amount for the same done work on multiple platforms.

One obvious solution was to normalize platform-specific metrics between different platforms. Almost immediately, it became obvious that this approach was impractical for several reasons:

1. Usable metrics would be limited to only the most common types (e.g. CPU time) which might not quantify the limiting resources on some platforms.
2. Even ubiquitous metrics might be difficult to normalize between dissimilar platforms (e.g. CPU time between RISC and CISC architectures).
3. The effort required to develop and maintain normalization routines would greatly exceed the viable maximum investment for an accounting system.

This course of investigation caused me to seek a source of platform-independent metrics which would be applicable across any platform. This got me to thinking about NBUs which are superior from an accounting standpoint, because they relate directly to the work the consumer is performing, and which are, by definition, platform-independent. In 1993, I began to conduct a seminar entitled, Accounting and Cost Allocation for Client/Server Systems, for Technology Transfer Institute, in which I recommended the use of NBUs as a basis for allocating information technology expenditures to consumers.

The Build versus Buy Question

Conceptually, the idea of using NBUs as accounting metrics was well-received, especially among the more financially-oriented audiences. There were exceptions, however, as in the case of a person who waited until the last hour of the seminar to ask, "You mean we should not use resource metrics for chargeback?" and when I agreed she insisted, "But everyone I know uses resource metrics for chargeback!" To which I replied "Lemmings all jump into the ocean, but that doesn't make it a good strategy for any individual lemming."

[1] *In the mainframe world, MVS is the predominant platform and all accounting methods are based around its metrics, standard data collections mechanisms (e.g. SMF), and commercial products which exploit them.*

Nevertheless, many enterprises were reluctant to embrace a strategy which required them to essentially develop their own metrics logging capability. I had to admit that this reluctance was well-founded because of the significant costs of developing a proprietary methodology and the potentially significant costs to convert that methodology to a standard approach if one became generally accepted. Therefore, I was pleased when Hewlett-Packard announced their Application Response Measurement (ARM) technology which provides a standardized means of capturing NBU counts.

What is ARM and
How Much Does it Cost?

First of all, ARM is not an accounting system! In fact, it is not a system (i.e. solution product) at all. It is an Applications Programming Interface (API) packaged with a software developer's kit (SDK) which allows applications developers to instrument their applications to collect NBU data. Specifically, according to the ARM API Guide, "The Application Response Measurement API is designed to instrument a unit of work such as a business transaction, that is performance sensitive. These transactions should be something that needs to be measured, monitored, and for which corrective action can be taken if the performance is deemed too slow."

From my point of view, the ARM designers missed the most important fact, that the business transaction counts themselves have intrinsic value in systems management. Nevertheless, they did provide a standard mechanism to instrument applications on most common distributed platforms. Moreover, both HP's Measureware and IBM's Tivoli system management products will collect and display ARM metrics.

Best of all, the ARM SDK is free! It comes on a CD-ROM which you can order through HP's web site (WWW.HP.COM) and includes a function call library and extensive code samples. The SDK contains libraries for UNIX (HP-UX, IBM AIX, NCR MP-RAS, and Sun Solaris), OS/2, Windows NT, Windows95, and Windows 3.1. Sample programs are provided in C/C^{++}, Pascal/Delphi, COBOL, and Microsoft Visual Basic.

Implementing ARM

To instrument an application with ARM, you need to insert six function calls at the appropriate

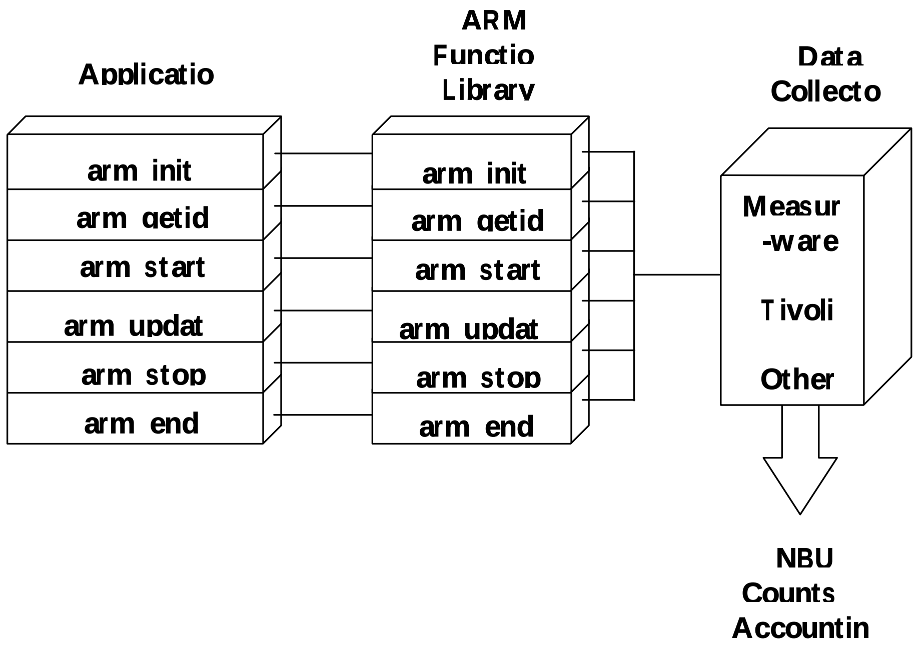

Figure 1. ARM Function Calls Within an Application

locations in its source code. These calls are:

1. **arm_init**
   Initializes the ARM environment for the application and return a unique application identifier which must be passed to arm_getid.
2. **arm_getid**
   Names a transaction and returns a unique transaction identifier to be passed to arm_start.
3. **arm_start**
   Defines the start of an individual transaction and returns a unique instance identifier which is passed to arm_update and/or arm_stop.
4. **arm_update**
   Optional call used to allow progress information to be recorded for long running transactions.
5. **arm_stop**
   Signals the end of an individual transaction.
6. **arm_end**
   Signals the end of the application and causes are to dismantle its measurement environment.

ARM instrumented applications must be distributed with a NOP (no operation) library which provides non-functional references for the ARM function calls. This library is replaced by a functional library which is unique to the specific measurement system used to capture ARM data.

With the functional library[2] in place, you simply need to input the NBU count data into your IT accounting system. Most IT accounting products have a generic interface which facilitates such inputs. The more feature-rich products allow prices to be specified for such data in the same manner as for standard metrics. Simpler products sometimes require the generic data to be pre-priced.

How Much to Charge for an NBU

Once you have instrumented an application to collect NBU data, you are still faced with the problem of how much to charge for each instance of a business transaction. This should be a lot more rigorous than most enterprises make it.

It is generally accepted, that the best decisions about how to allocate scarce resources are made by individuals who are most directly affected by those decisions. In a free enterprise society, those decisions are made by comparing cost to value. Thus the vendor’s price is the critical factor in the consumer’s allocation (purchase) decision.

If the enterprise’s IT consumers are going to make business decisions based on the amount of their IT charges, those charges had better reflect what it actually costs to provide a transaction. Similarly, when IT is an internal “vendor”, serving a semi-captive market, it should be able to explain what drives the cost of specific business transactions, how they differ from one another, and how technology changes will impact both the absolute and relative costs of key transactions. This level of financial understanding is a long way from what most IT producers know and dramatically more than can be gleaned from an IT services invoice.

Typical IT Pricing

Unfortunately, the focus of most IT service pricing methodologies is to arrive at prices which ensure that the total amount billed will approximate the total IT budget. More mature pricing strategies add the requirement that the consumption measures (metrics) employed be easy for IT to capture. And truly

[2] *Currently functional libraries are provided by HP to interface ARM to their Measureware products or by IBM/Tivoli to interface it to their products. In addition, their is sufficient information in the ARM Guide to allow you to write your own proprietary function calls*

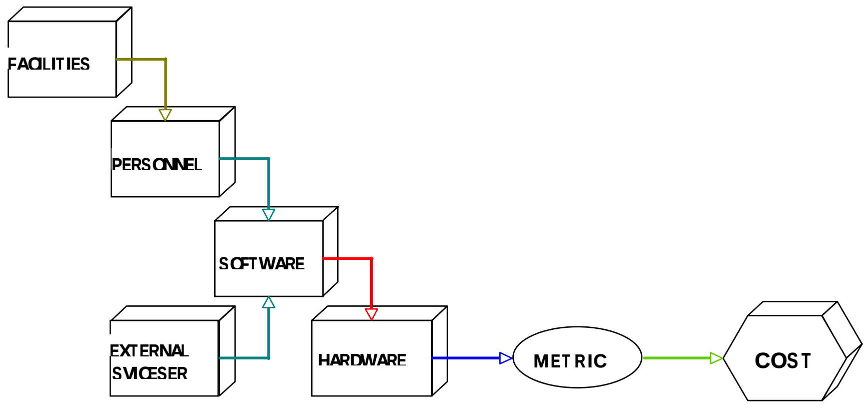

Figure 2. Typical Resource Pricing

sophisticated approaches attempt to choose metrics and prices which minimize the amount of complaining on the part of the customers.

This almost always results in an approach in which customers are charged for platform-specific hardware-based metrics, such as CPU-time, disk space used, number of input/output operations, number of lines printed, etc. These methods are generally termed resource accounting.

All IT operations, regardless of size, platform, or industry, consume five classes of resources. These are:

1. Hardware,
2. Software,
3. Personnel,
4. Facilities, and
5. Externally Purchased Services

With the exception of a sizable portion of the hardware and its directly associated software (e.g. CPU's and their operating systems), most of these resources can be easily allocated to specific business systems. Unfortunately, resource accounting methods usually burden the majority of these costs on the most basic common resource, generally CPU-time. This approach is shown in Figure.

As you might expect, there is little likelihood of this approach producing prices which bear any relationship to the actual cost of a specific service. Consumer understanding (which is closely related to acceptance) also suffers, because to explain the price you first have to explain what an obtuse, platform-specific, technical measurement means.

NBU's and Resource Pricing
an unlikely marriage

Enterprises attempting to migrate from strictly resource-based to transaction or NBU-based pricing, often attempt to profile the target element. They calculate how many CPU-seconds it consumes, how much disk it uses, and so forth. This allows them to calculate a price, but it is typically no closer to actual cost and not much easier to explain.

There is a better approach, however, one which incorporates the direct allocation of cost elements to business transaction. This approach is called ***Activity-based Costing*** or ABC. ABC was first proposed in the mid-1950's as a way to address costing in the then-emerging services sector. However, it didn't become a buzzword until the mid-1980's when Robert S. Kaplan of the Harvard Business School published his widely-acclaimed book *Relevance Lost, The Rise and Fall of Management Accounting*.

Using the ABC methodology, you first identify the things which an organization does (the *Activities*) along with their inputs and outputs. Second, you determine how each activity consumes costs by analyzing its consumption of materials, labor, services, etc. Finally, you determine the sequence of activities which leads from a business transaction (NBU) back to the raw materials and labor. With this information you can accurately allocate actual costs to individual NBUs. There is only one problem with ABC, it is extremely expensive and time-consuming to implement!

It seems like there should be a way to get most of the benefits of conventional activity-based costing approaches at a price that most IT organizations can afford and there is, it is called *Delivery-Level Architecture*!

Delivery-Level Architecture

The Delivery-Level Architecture (DLA) is a framework for activity-based cost allocation which addresses the specific requirements of IT service providers. The DLA approach simplifies the development of an ABC cost model by using the unique characteristics of IT service delivery to structure the definitions of activities. This structure or framework greatly simplifies the process of identifying activities and their associated inputs and outputs with the benefit that the time and effort required to implement a DLA-based cost model are greatly reduced.

It explicitly quantifies the differences between basic resource costs and delivered (i.e. application) resource costs. The Delivery-level approach (see Figure 3) views the IT services provider as a multi-level value chain. Each level adds both value and cost to the IT service. Each level utilizes the output of the level below it and adds both cost and value. Lower levels of the hierarchy provide more generic services while higher levels are more specialized.

In essence, DLA is a multi-level activity-based costing model. The general structure of the model can be applied to any IT organization regardless of platform, type of applications or implementing technology. A specific implementation of the model can be tailored to a particular enterprise or site by identifying particular applications at the highest level and selecting only the applicable technologies at the lower levels. DLA is generally applicable to all platforms and technologies, making it particularly effective for diverse environments.

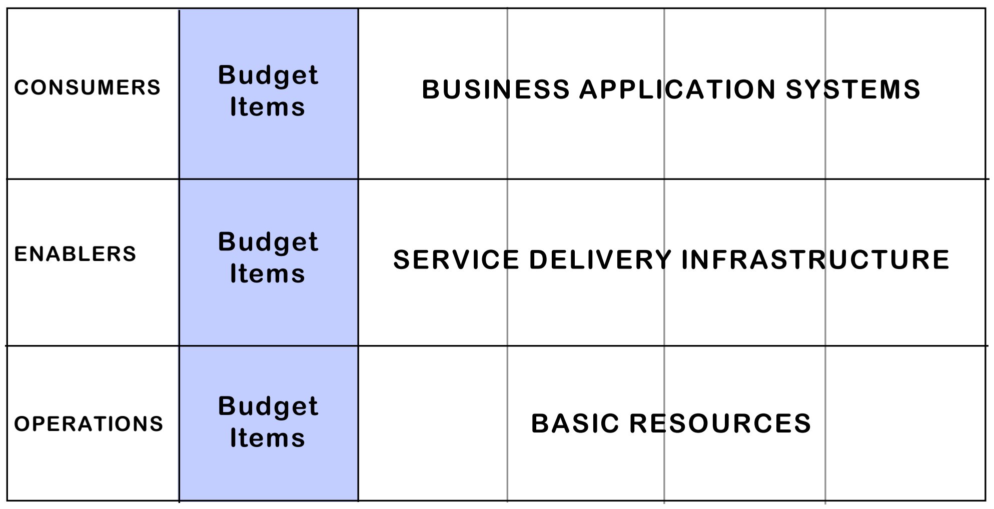

Figure 3. Delivery-Level Architecture Schematic

Structure of the Delivery-Level Architecture

The basic Delivery-Level Architecture is a two-dimensional grid with three rows and three columns (see Figure). Each row represents a step (layer) in the IT delivery process. The lowest level, Operations, provides basic resources such as CPU cycles and “raw” storage. The intermediate level, Enablers, provides functional environments such as batch, TP monitors, and managed storage. The highest level, Applications, provides individual business application systems which add value to the business process itself. The left-hand column is used for labels only. The middle or “Costs” column contains the budgeted cost elements which are specific to each level. Note that only a few costs are shown but, in practice, this column would contain the entire IT budget. The right-hand or “Targets” column contains the cost pools which are specific to each level.

Operational Costs

This level represents the costs (middle column) and “products” (right column) provided by the lowest level of the IT delivery process. These are essentially the raw materials of information technology and are consumed to produce every other IT service. From a cost standpoint, this level is dominated by hardware, operating systems software, and facilities costs. While some personnel costs are included at this level, they are limited to the most basic systems and operations support. The allocation targets are the fundamental technical resources of IT: processing, data access, data storage, communications and output. Because only the most basic costs are included at this level, the cost of a resource is the same regardless of how it is subsequently used (i.e. CPU cost is a function only of platform and quantity).

Enabling Costs

This level of the architecture quantifies the costs which are incurred to transform the basic resources into specific technical services such as: databases or groupware. Generally, these services are designed to support multiple simultaneous users, so the basic resources they consume must be apportioned uniquely based on how the service functions. The costs associated with each service are the software itself and the labor required to install, operate and maintain it. These costs may differ greatly between service classes, individual products within a class, and individual enterprise implementations.

Application Costs

This level of the architecture encapsulates the costs associated with providing specific, tailored services to individual technology consumers. These services are embodied in the business application systems (e.g. general ledger, order entry, etc.). These applications may either be in-house developed or purchased from an external vendor. For in-house developed applications, the majority of the costs result from developer’s salaries, their tools, and IT usage costs resulting from development and maintenance. For purchased applications, the major costs are purchase price, maintenance fees and personnel costs related to installation tailoring and maintenance.

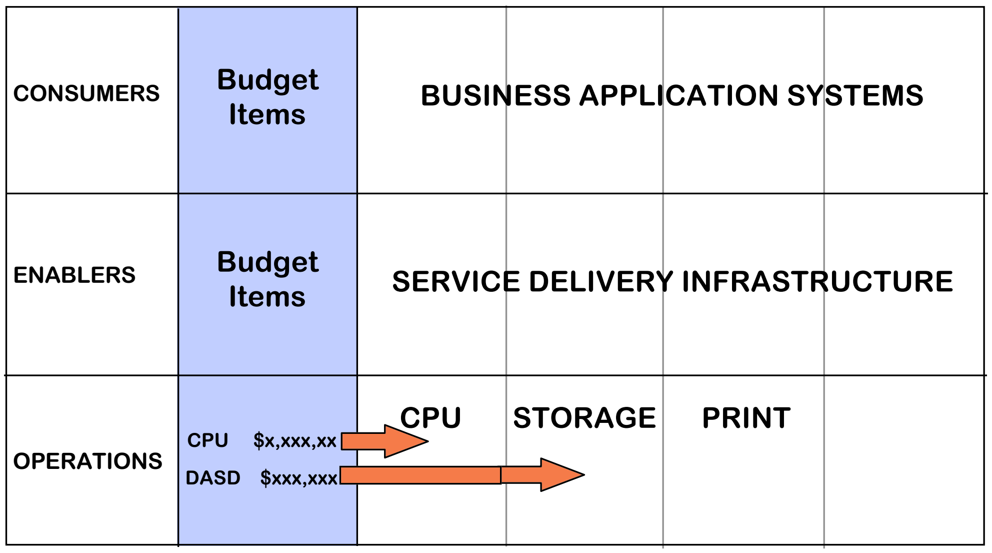

Figure 4. Horizontal Allocation in DLA

Allocating Costs with DLA

Allocating costs via the DLA methodology is a three-step process. First, basic or "raw" costs, typically obtained from the IT operating budget, are assigned to the DLA level where they are consumed. This process is termed *stratification*. Second, the costs are assigned to one or more cost pools within their DLA level. This is call *horizontal allocation*. Finally, the cost pools in each DLA layer are burdened with the costs from the layer below. This is called *vertical allocation*. Each of these steps is explained more fully below.

Cost Stratification

Cost Stratification starts by defining the cost pools for each DLA level. Starting at the lowest or Operations level, functional hardware is divided into major categories such as processing, storage, communications circuits, etc. Direct costs are then associated with these pools. For example, processor hardware is assigned to a processing cost pool along with its associated operating system and direct support personnel.

Once the Operations layer costs have been assigned, the Enablers layer cost pools are defined based on major processing environments such as databases, TP monitors, groupware, etc. Again direct software and personnel costs are assigned to the appropriate pools along with any dedicated hardware costs.

Cost pools for the Applications layer are defined by the enterprise's major applications and their associated NBU's. In some cases, a well-defined NBU is the product of more than one application; in those cases group those applications together.

As a general "rule-of-thumb", most of the hardware will be assigned to the Operations layer. The Enablers layer will typically be mostly software and personnel. And the Applications layer will have the smallest direct costs, again being primarily software and personnel.

Horizontal Allocation

The distinction between stratification and horizontal allocation is obvious in theory and almost invisible in practice. Theoretically, costs are allocated horizontally from the costs column to the appropriate cost pools (targets) within a row as shown in Figure . The costs are spread to the target cells based on allocation rules (algorithms) which are implied but not shown on the diagram. Because the costs have already been segregated by delivery level, 100% of an individual cost item is allocated to a single pool in the majority of cases.

In practice, the process of stratifying costs and defining cost pools is an iterative one. Costs may be grouped and regrouped several times until the allocation team finds a structure that "works". Nevertheless, this process tend to be fairly intuitive and is accomplished relatively easily.

Vertical Allocation

Once horizontal allocation is completed, costs are allocated vertically from each lower level to the level above it. This vertical allocation process quantifies the value-chain which leads from the "raw" IT resources to the business element which is enabled. Lower level cost pools are allocated to pools in the level above based on a metric which is employed as a consumption proxy3 or cost driver. These metrics are accumulated based on the characterization (workload grouping) for the target cell. This allocates the source (lower row) cost pool on the basis of its usage.

In the example shown in Figure 1, the "raw" cost of processing is shown at the Operational level measured in units of CPU time. This cost consists of the cost of processor hardware, operating systems, basic systems administration and operation. Many enablers utilize the CPU, so its cost must be apportioned to each, depending on which ones the physical processor supports. Thus CPU time costs are burdened (added to) databases, TP monitors, groupware, etc. in the proportion that each enabler consumes CPU time. The workload groupings used to characterize CPU time vary based on the specific enabler.

The costs of the Enablers (in the example Database and TP Monitor), including both their direct costs, and the cost burden on them from the Operations layer, are allocated to the applications based on transaction counts. Note that in this case, the database transactions and the TP monitor transaction represent different measurements.

Coming Full Circle

At the top of the DLA hierarchy, all of the enterprise's IT costs have been allocated to specific applications. The cost of these applications can then be allocated to their users based on actual counts of business transactions (NBUs). How are these NBUs counted? By instrumenting the applications with ARM or some similar mechanism.

Value of the Delivery-Level Architecture

Successful cost allocation methodologies for information technology must provide acceptable answers to a number of questions:

- What are the appropriate allocation "targets" (cost objects)?
- Do resource costs differ depending on use?
- What "drives" IT costs and how could those costs be reduced?

The Delivery-level Architecture provides a practical means of characterizing costs in the technology consumer's (end-user's) terms. Because it allows costs to be captured at the level of individual applications, it also facilitates pricing services in terms of the business transactions which are supported by those applications. This has the effect of aligning IT expenditures with the business benefits which those expenditures enable.

[3] The terminology commonly used to represent allocation metrics differs between financial management and IT systems management. In this paper, the term proxy is used to indicate any type of metric. The term metric is reserved for technical measures provided by a software monitor. Cost driver is the term commonly used in financial literature for allocation proxies used in activity-based costing methodologies.

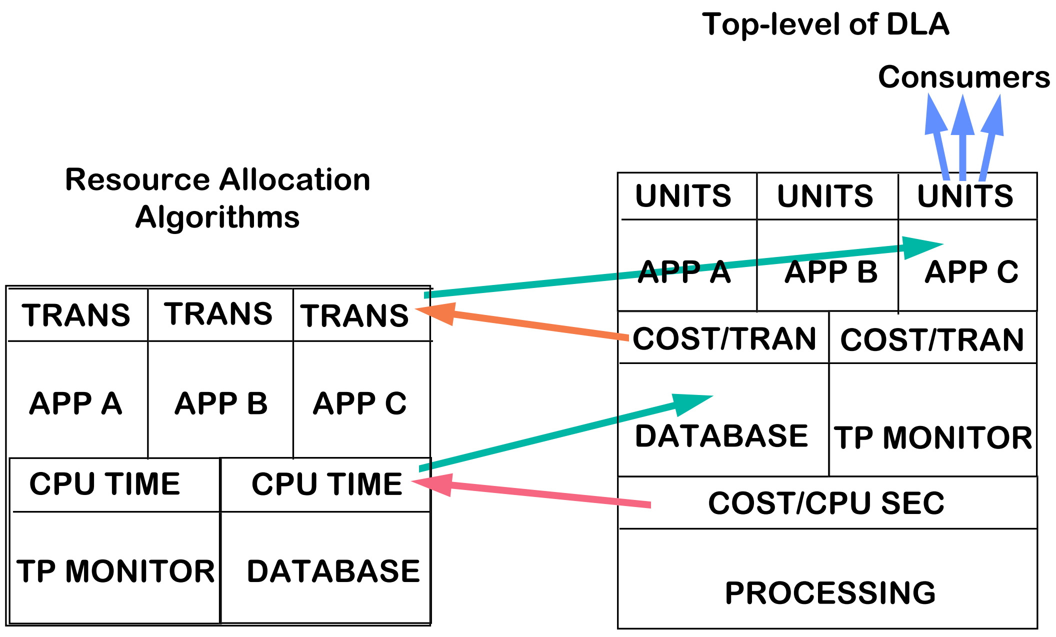

Figure 1. Vertical Allocation in DLA

The Delivery-level Architecture shows that basic resources (e.g. CPU, Storage, etc.) have the same cost regardless of how they are delivered or what application utilizes them. This is an obvious fact which is generally obscured by traditional cost allocation schemes. More importantly, it also shows why different enablers cost more and therefore how the unit costs aggregate to specific applications. This allows for cost comparisons at each level of the value chain.

The activity-based cost allocation provided by DLA facilitates cost reduction and cost containment efforts by localizing expenditures to the activities which actually incur them. Simply knowing which stages in the production of a particular service are the most expensive serves to focus cost management efforts where they have the greatest possibility of impact.

The Architecture also supports multiple cost allocation approaches. Pricing can be done at the application level by summarizing the allocations to individual business transactions. Prices can be set at lower levels of the architecture by simply burdening costs from the layers above evenly on the selected level's metrics. Of course, significantly less cost management information is available at lower levels of the architecture.

Summary

Used together, ARM and DLA represent a substantial improvement in IT financial management capability which can be implemented at a cost virtually any enterprise can afford. Very good DLA cost models have been built by large IT providers in less than six calendar months with as few as two dedicated full-time employees. The cost to implement ARM depends on the number of applications involved, but most organizations find that the number of critical business transaction for which they wish to charge is relatively small. However, the implementation cost, whatever it may be, is insignificant when you realize that NBU-based chargeback and DLA-based costing can provide the reality of IT financial management, rather than simply the illusion.