Activity Based Costing for Component-Based Software Development

ROBERT G. FICHMAN   fichman@bc.edu
Boston College, Wallace E. Carroll School of Management, 452B Fulton Hall,
140 Commonwealth Ave. Chestnut Hill, MA 02467-3808, USA

CHRIS F. KEMERER   ckemerer@katz.pitt.edu
University of Pittsburgh, 278a Mervis Hall, Pittsburgh, PA 15260, USA

Abstract. Component-based software development is a promising set of technologies designed to move software creation from its current, labor-intensive, craft-like approach to a more modern, reuse-centered style. However, a lesson learned from previous radical software process innovations is that a strong technology alone is generally insufficient for successful adoption. In order for gains to be realized from such technologies the management practices surrounding the implementation of the new technology must also change. It is with this view that we propose the adoption of a complementary management approach called activity based costing (ABC) to allow organizations to properly account for and recognize the gains from a component-based approach. ABC enables a management environment where appropriate incentives are created for the development and reuse of software components. Data from a large software vendor who has experience with ABC in a traditional software development environment are presented, along with a chart of accounts for a modern, component-based model.

Keywords: activity based costing, ABC, activity based management, ABM, component-based software development, object orientation, objects, expert services, software factory

1. Introduction

The nature of software development is undergoing a dramatic change, one that strikes at the core of how the industry will build software systems in the coming decade. Traditionally, large systems have been custom-built, using a "waterfall" approach to development. Under this approach, a project begins with a requirements or needs analysis, then proceeds linearly through stages of design, coding, testing, documentation, training and implementation.

This custom-crafted approach to software is very expensive, as each project develops a custom product. This, despite the fact that many applications contain numerous opportunities to rely on work products created for previous systems, including not only program code, but analysis and design artifacts as well. While this issue has been widely recognized, until recently modern software technologies made re-use possible on only an ad hoc or local basis. Large-scale component reuse was typically not accomplished.

In response to this, a new approach has been proposed for the development of these systems. Using object-oriented or other modern development tools many observers have

proposed what amounts to a manufacturing process for software development [5,6]. This enhanced delivery capability should simultaneously result in cost savings and a higher level of quality, as this approach allows for the reuse of proven components [21].

However, this new approach to software delivery creates a significant number of managerial challenges. The development of reusable components for later assembly into potentially many systems is a radical departure from the current, single-use, project based approach that predominates in today's systems development activities [19,20,22,25]. Many barriers stand in the way of adoption of this new approach, including: an absence of technical infrastructures to support component reuse; an absence of managerial guidelines for running component reuse projects; uncertainties about the best ways to structure the work of reuse; lack of standard approaches for how to handle ownership, coordination and versioning issues related to reusable components; and issues related to reuse economics, costing and incentives [10,12,15]. Of these, we believe that barriers related to economics, costing and incentives will figure especially prominently in the ultimate level of adoption of large-scale reuse because these barriers are not likely to be addressed through the ordinary course of technological progress in reuse methods and tools.

In a typical software development environment, project managers have very limited incentives for investments in reuse, and, in fact, often have strong disincentives, in terms of the likely negative impacts on project budgets and schedules that investments in reusability often entail [26]. An approach that emphasizes the development of reusable components, on the other hand, will require strong management of incentives for component development, perhaps even to the point of creating separate reuse teams [3,9,15]. This, in turn, raises critical issues for project costing. In the current environment of single use, the cost of software is simply the cost of the direct labor hours that went into its development. With multiple use, this direct relationship breaks down because an organization must devote considerable resources to reuse activities (e.g., maintaining reuse infrastructures, developing and maintaining components, locating and evaluating components, adapting and integrating components) that may be performed by – or on behalf of – other project teams [24] or may occur outside of traditional project teams altogether [3].

Given that when component-based software development is discussed an analogy is often drawn with hardware manufacturing, it is not surprising that manufacturing should also be the source for tools to assist with some of the management challenges offered by this new approach. For guidance in how to manage issues related to costing and cost management we look to work that goes under the rubric of "activity based costing," or ABC [4,17,23]. This is an approach originally designed as way to improve the allocation of overhead costs in manufacturing settings in order to develop more accurate product costs. With this approach, overhead and indirect costs are allocated to products based on the degree to which those products actually "consume" the repetitive production activities of the firm, rather than being based on a single factor, such as the number of direct labor hours associated with the product. Conventional methods of manufacturing product costing have relied on assumptions about the labor-intensive manner in which products

were developed years ago, assumptions which have changed with increasing automation and which today can produce startling anomalies in actual vs. standard costs [16, ch. 1, 23]. Similarly, the introduction of large-scale component reuse creates the potential for a disconnect between the results of current software costing methods and the actual costs involved with a reuse-intensive software process, because it likewise decreases the dominance of direct labor costs in the total costs of software products. Without ABC or some other new approach to costing, the substantial costs of developing reusable components and maintaining an environment conducive to reuse will be misallocated. If the project teams themselves perform the activities related to reuse, then the costs will end up hidden in the direct costs of different projects, and the project teams will operate under major disincentives to invest in components for subsequent reuse. If, on the other hand, separate reuse teams are created, there will be no way to properly allocate what could be very substantial overhead costs associated with reuse teams back to individual project teams.

As a result it appears that the application of ABC principles to component-based software development could lead to a greater understanding of this new process of developing software. Such an understanding should eventually lead to:

- reduced cost of software development, as costly activities are identified and rationalized;
- greater control over delivery dates and schedules;
- reduced risk of budget over-runs;
- improved alignment of incentives related to reuse;
- more effective pricing of systems and system components.

The remainder of this article is organized as follows. Section 2 introduces the ABC concept and applies it to the case of component software development. Sections 3 and 4 then describe research work with and descriptive empirical data from a major commercial software vendor that has applied ABC to a traditional software development process. Section 5 then extends this analysis to develop a proposed set of ABC accounts for a component-based software development organization. Section 6 provides a brief summary and conclusions.

2. ABC for software

In a systematic component reuse environment, indirect costs and overhead associated with reuse can become a significant portion of software development costs, possibly exceeding 50% [3]. Traditional methods of accounting for indirect costs and overhead (e.g., based on direct labor hours) are not an effective means of allocating costs in such an environment, as these methods will tend to under-cost projects that are comparatively heavy consumers of reuse activities, and over-cost projects that are light consumers of reuse activities, thus leading to inappropriate conclusions about the true cost of those projects. To see why, let us suppose we have two projects, both with 5000 h of direct

labor and an overhead allocation for reuse support of $30 per direct labor hour. This overhead allocation is required because this hypothetical organization has developed a large team of dedicated reuse specialists tasked with assuming most of the burden of developing reusable components and making them available to project teams, as per the "software factory" model of reuse [3]. These specialists analyze software domains for reusability, develop and document reusable components, assist project teams in assessing reuse opportunities, find and adapt components for use by project teams, certify and document components offered by project teams and many other activities (as will be described in section 5 in the ABC chart of accounts for software reuse). None of these specialists is actually a part of either project team, whose members are tasked only with developing software for their own project's application. Now, assume that the first project had little opportunity for reuse, and, in fact incorporated only one half the average percent of reusable components as part of the final work product, while the second project was a heavy consumer of reuse with double the average percentage of the final work product being deriving from reused components. Despite this, each project would be charged $150,000 (5000 × $30) to allocate the reuse overhead activities. In this example, the net result of using traditional overhead allocation methods would be an overcosting of the first project on the order of $75,000, and undercosting of the second project by about $150,000. With ABC, by contrast, the indirect and overhead costs of reuse would be apportioned based on the extent to which these projects actually "consumed" the reuse related activities performed by the reuse specialist team, with the result being much more accurate project costing.

Furthermore, traditional methods provide no systematic means for assessing and improving the efficiency and effectiveness of reuse activities. ABC methods and associated Activity Based Management (ABM) principles provide a means to address both of these shortcomings of traditional approaches.

ABC and ABM were originally developed to address product cost anomalies in a manufacturing environment [16,17,23]. Beginning in the 1980's it was observed that indirect costs and overhead were comprising an ever growing proportion of true product costs, yet traditional product volume-based cost allocation methods did not properly distribute those costs. The result was that relatively complex, low volume products, which were bigger consumers of overhead on a per unit basis, were under-costed, while high volume, simple products were over-costed. Such costing anomalies can lead to poor decisions in any domain where accurate knowledge of product costs is key, such as for product pricing and product mix decisions.

ABC was developed to provide a more accurate approach to allocating overhead costs to products. The basic insight underlying the ABC approach is that the repetitive activities employed in the actual work of an organization can provide a sounder basis for allocating costs. With the ABC approach, rather than viewing work products as being direct consumers of overhead resources, the repetitive activities required to actually produce products are introduced to stand in between products and costs. These activities are viewed as the consumers of resources, and in turn, products are viewed as consuming

activities. ABC provides a systematic method for mapping resource costs to activities and from activities to products.

Although ABC was originally developed for repetitive manufacturing environments, it has since been generalized as a means to map true resources costs to any *cost object* of interest not just products, but also services, customers, branch offices, and projects [17, ch. 12, 27, p. 85]. However, ABC does not apply to every situation. For ABC to apply well, three conditions should be met:

(1) indirect and overhead costs must be significant, and must be poorly accounted for by traditional means;

(2) objects must exist for which management wishes to know true costs (products, customers, projects, etc.);

(3) repetitive activities must exist that can serve as the basis for mapping indirect and overhead costs to cost objects.

The first two criteria go to the relative pay-off from ABC, i.e., the pay-off will be highest to the extent that these two criteria hold [8,17, pp. 100–101]. The third criterion is self-evident, since the whole point of ABC is to use repetitive activities to better account for the costs associated with cost objects.

A systematic reuse environment that relies on component-based software meets these three criteria. First, such environments are often characterized by high overhead costs associated with centralized reuse teams and infrastructure [3,19], and these costs need not be highly correlated with direct project hours or costs because the amount of reuse can vary widely from project to project [11]. Second, management typically is quite concerned with the cost of software development projects and/or software products. For example, at ComputerCo (the site of our empirical work) one major new product was priced very aggressively based on the observation that, since reuse was substantial on the project, it would be possible to have a high return on investment even with aggressive pricing. Third, repetitive reuse related activities do exist (as will be explained below) that can serve as a mapping between overhead costs and projects (or other cost objects).

ABC can provide two kinds of benefits to organizations practicing systematic reuse. First, because it supports more accurate allocation of overhead costs, it can lead to more accurate conclusions about the cost effectiveness of individual projects. In those instances where the result of a project is a product sold to others, it can provide the basis for pricing and for judging profitability.

Second, and perhaps equally important, ABC can provide the basis for assessing and improving the productivity and effectiveness of reuse activities. Prior work modeling the economics of reuse provides a means to predict and judge whether the benefits of systematic reuse can be expected to outweigh the costs in the aggregate [13]. However, this work has not been intended to provide guidance for how to actually lower the costs of reuse for any given level of benefit. In today's business environment, management has developed a low tolerance for overhead work even when it does, in fact, contribute to the health and profitability of the organization. We believe this low tolerance stems,

Table 1
ABC terminology (adapted from [27]).

Term	Definition
Activity-based costing	A method of measuring the cost and performance of activities and cost objects. Assigns costs to activities based on their use of resources, and assigns costs to cost objects based on their use of activities.
Cost assignment view	Shows how costs are assigned to cost objects.
Resources	Economic elements applied or used in the performance of activities (e.g., labor, equipment, materials, and floor space).
Resource cost driver	The link between resources and activities. Determines an activity's unit cost.
Activity	A repetitive unit of work performed within an organization.
Activity cost driver	A factor that measures the extent of activity consumption by a given cost object.
Cost object	The reason for performing an activity. Cost objects include products, services, customers, projects and contracts.
Process view	Provides operational information about an activity.
Causal cost driver	Determines the overall effort required of an activity and the resources needed.
Performance measure	An indicator of the work performed and the results achieved in an activity.

at least in part, from a perception that overhead work tends to be performed inefficiently, and has no clear mapping to the value producing activities of the organization. A well run ABC/ABM program can counter this perception by showing where overhead costs come from and where they go, and can also counter the possible reality underlying this perception by providing a means to assess and improve the performance of activities that are, in fact, heavy consumers of overhead resources.

We use the term *ABC Model* to refer to the overall structure of activities, resources, resource cost drivers, cost objects, etc. comprising an ABC implementation. In this paper we have chosen to standardize on the view of ABC articulated by Turney [27] (see table 1).[1]

ABC takes two alternative views of the activities performed in an organization, as illustrated in figure 1 below. This diagram, developed by Turney [27], was first presented to the Consortium for Advanced Manufacturers-International, and thus is commonly referred to as the "CAM-I Cross" [4, p. 54]. The first of the two views is called the *cost assignment view*. This view (depicted vertically in figure 1) shows the flow of costs through an activity, and consists of *resources, resource cost drivers*, *activities*, *activity cost drivers* and *cost objects*. The second is called the *process view*. This view (depicted horizontally in figure 1) shows the flow of performance-related information through the

---

[1] However, to better conform to emerging consensus on ABC terminology, we use the terms *resource cost driver*, *activity cost driver*, and *causal cost driver*, in place of the terms originally used by Turney, i.e., *resource driver*, *activity driver*, and *cost driver*, respectively.
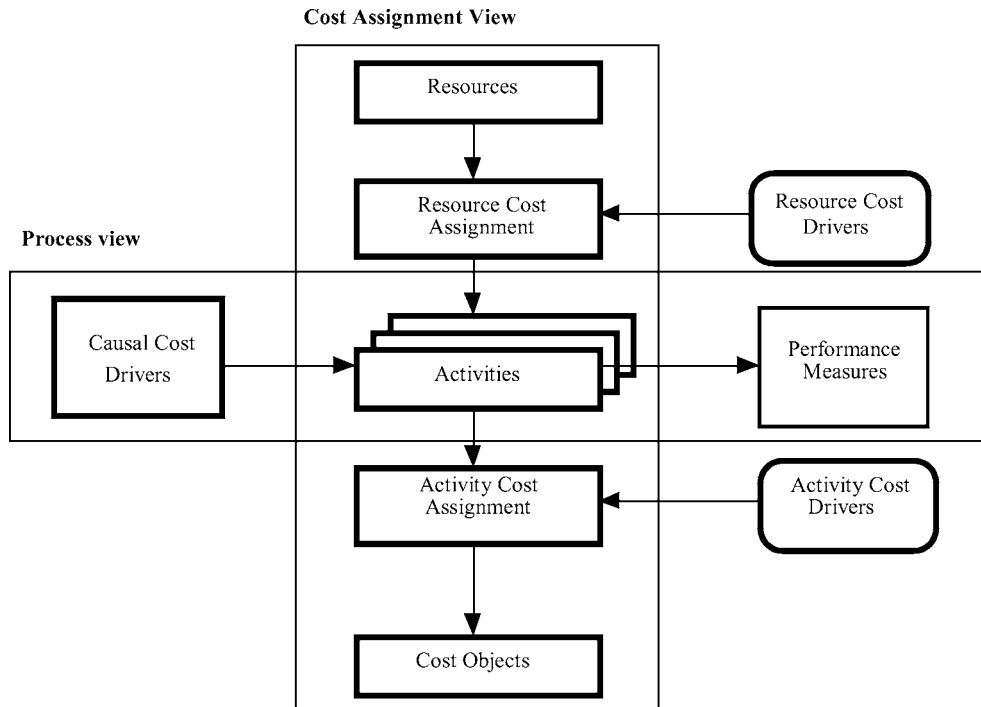
Cost Assignment View

Figure 1. Structure of an ABC model [27].

activity, and consists of *causal cost drivers*, *activities*, and *performance measures*. The purpose of the first view is to support more accurate allocation of costs; the purpose of the second view is to guide cost reduction efforts. (Definitions of all highlighted terms are provided in table 1.)

The cost assignment view of an ABC model is developed according to the following seven step procedure (see [4,27] for more detailed guidance):

(1) identify the repetitive activities within the scope of the ABC effort. This is often referred to as developing the Activity Dictionary [17, ch. 8];

(2) group the activities into activity centers (collections of activities that share the same overhead cost pools);

(3) identify the resources used in overhead work and aggregate them in to pools by activity center;

(4) identify resource cost drivers, e.g., drivers that determine the relative share of each overhead cost pool attributable to each activity;

(5) determine the basis for calculating unit costs of each activity;

(6) identify cost objects, e.g., products, customers, products;

(7) determine the activity cost drivers, i.e., drivers that define the extent of consumption of each activity by cost objects of interest.

The process view of an ABC model is developed according to the following two step procedure:

(1) identify causal cost drivers, i.e., what drives the level or volume of each activity;

(2) identify performance measures, i.e., measures that can be used to determine how efficiently and effectively each activity is performed.

Once the structure of an ABC model has been defined, the next step is to set aside a period of time for collection of data about the activities, and to use these data to determine standard values for resource cost drivers and activity cost drivers to be used in cost allocation.

3. Research approach

To examine the feasibility of this approach, we worked closely with a major software developer who had a number of recently completed projects that could be evaluated using an ABC approach. Through a series of on-site interviews and review of contemporaneous documents, we documented a set of completed projects using a prototype ABC approach. These results are presented below. Based on this experience and a survey of the reuse literature, an ABC model and chart of accounts were further developed for a future component-based software development approach.

3.1. Methodology

The primary research approach was to investigate a set of field based cases of a major software developer, which we will refer to as ComputerCo. The selection criteria for the study favored projects thought to represent successful current examples of reuse and those that were using object-oriented tools and methods. A total of fifteen projects were included in the study, of which four are highlighted here merely to illustrate various approaches. The primary data collection approach was a series of semistructured interviews with project managers and team members from a set of completed or nearly completed object-oriented projects. The focus in these interviews was to capture project management differences developed in the field to aid management of these modern projects. More structured data were also gathered using a standard questionnaire. Altogether we gathered data on staffing, project size, activities performed, allocation of budgeted costs by activity, software development technologies used, the nature of the software being developed, extent and nature of software reuse achieved, budget and schedule conformance, perceived reuse facilitators and inhibitors, and details regarding how reuse was managed (processes, tools, measures, economic models, funding mechanisms, etc.). Of this, the data related to activity costs are most pertinent to the present study.

4. Results

4.1. Prior and new management models compared

A natural starting point for thinking about ABC analysis for software is in the work breakdown structures (WBS) related to specific systems development activities in the lifecycle. To this end our data-site, a developer of packaged software, created a list of 27 activities to track. This list is shown as table 2. A description of each appears in appendix A.

Note that on the list there are many traditional overhead activities that are necessary functions for the sale and support of software products. These overhead activities (most notably activities 20, and 24 through 27) need to be properly allocated to products. Therefore, they would represent candidate activities to be analyzed using an ABC approach. In addition, there are many "overhead" style activities relating to software development activities (including activities 5, 17, 18, 21). The development of this type of detailed work breakdown structure is a useful first step for software development organizations who wish to better manage their costs, whether or not they perceive a need to allocate these costs appropriately to software products.

Special note might be taken of Activity #5, Reuse Planning and Acquisition. While in part addressing how the current product might reuse existing components, it also investigates how components developed for this product might be created in such a way as to increase the opportunities for their reuse in other products. In a traditional software development project, this may be all of the activity devoted toward reuse. Note that for these four examples (and for the set of 15 projects generally) the amount of effort devoted to this task was relatively small. This is consistent with much of the literature on software reuse in terms of the limited amount of reuse that often takes place when projects are managed in the traditional way [19,22].

In examining the data for these earlier projects, it is clear that they differ markedly in their use of resources. Product 1, a relatively older and largely traditional-style project, had a considerable amount of time in Activity #2 Plan And Project Management and Activity #6, Code And Unit Test, versus later projects. This is a product that was managed in what might be termed a "project style", rather than a "product style" approach, whereby the focus is on planning and executing this project in relative isolation to other, either predecessor or successor projects. Product 2, of similar vintage as Product 1, differs mainly in how resources were devoted to design activities rather than testing activities. One use of such data may be in determining the relative efficiency of such differing strategies. For example, projects with relatively higher percentages of resources in the design phase could be analyzed to determine whether they had consistently lower than average costs for other activities (such as testing), or for subsequent projects (such as to develop later versions of the product). Project 3, a newer vintage project, demonstrates how a significant amount of resources can be devoted to overhead activities (e.g., 39% in Activity #24–27, SG&A) in software development, and therefore how critical to managerial decision making it can be to properly account for overhead costs. Finally, Product 4, a recently completed product, reflects a somewhat different pattern, even

Table 2
WBS distributions for traditional projects.[a]

	Product 1%s	Product 2%s	Product 3%s	Product 4%s
Systems development and maintenance				
1. FUNCTIONAL REQUIREMENTS ANALYSIS	3	2	1	5
2. PLAN AND PROJECT MANAGEMENT	13	2	1	5
3. PRODUCT DESIGN	0	2	3	10
4. INTERNAL DESIGN	0	8	2	10
5. REUSE PLANNING AND ACQUISITION	0	0	1	2
6. CODE AND UNIT TEST	38	23	19	25
7. OTHER PROGRAMMING SUPPORT	8	0	1	1
8. TEST PLANNING	3	2	3	2
9. COMPONENT TEST AND PRODUCT VERIFICATION TEST	4	0	2	8
10. SYSTEMS TEST	9	3	2	10
11. PERFORMANCE ANALYSIS AND MEASUREMENT	2	1	1	3
12. OTHER VERIFICATION AND VALIDATION ACTIVITIES	0	1	1	2
13. CONFIGURATION MANAGEMENT AND QUALITY ASSURANCE	1	2	0	1
14. INFORMATION DEVELOPMENT	5	5	3	5
15. MAINTENANCE, Direct Support of Customers	0	10	2	NA
16. MAINTENANCE, Fix Correction				
17. MAINTENANCE, Support Coordination	13	17	3	NA
18. MAINTAINING DEVELOPER WORKSTATION	0	0	2	1
Program-products management and planning				
19. PROJECT OFFICE FUNCTIONS	0	0	0	2
20. PRODUCT AND MARKET PLANNING	0	3	7	1
Program supervision and general				
21. COMMON GENERAL DEVELOPMENT ENVIRONMENT/TOOLS AND SUPPORT	0	1	0	0
22. MANAGEMENT SUPERVISION	0	5	4	3
23. ADMINISTRATIVE SUPPORT	1	4	3	2
Sales, general and administrative	0	10	39	
24. MARKETING SUPPORT				0
25. DISTRIBUTION				1
26. MARKETING COMMUNICATION				0
27. MARKETING DEVELOPMENT				0

[a] Disguised data. Cells show the budgeted cost of the activity as a percentage of the total budget for the project. Some columns may not add to 100% due to rounding. A value of zero represents a true value between 0.0 and 0.4. Some prior projects not measured at lowest levels of detail.

when allowances are made for the fact that full maintenance has not yet begun. This product was developed using object-oriented techniques, and the distribution of activi-

ties devoted to early stage analysis, reuse planning, and component testing, was higher than with earlier style products. Given the relative newness of this technology within the organization, these figures may be a conservative representation of the changes in software process resource allocations, as it might be expected that the development and management techniques are likely to remain anchored in earlier practice, and therefore may lag the surface technology change [7]. As these brief examples show, there appears to be a clear opportunity to apply ABC concepts in a conventional software development setting. This suggests that in the case of a systematic component reuse environment the potential benefits of ABC thinking will be even greater.

5. A proposed ABC chart of accounts for a component-based reuse organization

The first step in developing an ABC model is to identify the repetitive work activities within the area of interest. Based on the reuse literature together with our examination of high reusing projects at ComputerCo we identified thirteen systematic reuse activities, which we grouped into six activity centers (see table 3).[2] We do not intend these to be a definitive list of reuse activities; each organization would, of course, be expected to develop their own activity dictionary based on how their own local processes of reuse are established. However, we do believe that these thirteen activities could prove to be reasonably generalizable, and will at least provide an advanced starting point for an organization that wishes to implement ABC for this purpose.

The above model is based on five key assumptions about the nature of reuse and the goals for ABC modeling. These assumptions arose out of our observation of what has worked – and what has not worked – regarding software reuse on the fifteen projects we examined [10]. Our first assumption is that most of the work of reuse is performed outside of conventional project teams, as will be the case under, for example, the Expert Services or Factory Models [14,15]. This assumption arises from the fact that this was the original vision for reuse at ComputerCo, and it is also the most demanding environment from a cost recovery standpoint. Where this is not the case, such as when most of the work of reuse is performed by project teams themselves, a simpler, and more focused ABC model may be more appropriate, although it will be necessary to adopt some other means for managing incentives regarding development of reusable components.

Our second assumption is that the target organization actually performs these activities in a systematic and measurable fashion. An organization that does not perform these activities in systematic and measurable fashion will require a different model, and may not be able to use ABC at all.

Our third assumption is that the primary cost objects of interest to the target organization are software development projects that consume components from shared li-

---

[2] Our primary source in the literature for identifying reuse activities was [14] although we also obtained insights into activities 1–4 from Wasmund [28] and into activity 5–11 from Frakes and Terry [11]. (Kim and Stohr [19] also provide a nice description of reuse activities that differs slightly from the ones presented here.)

Table 3
Reuse activities.[a]

Activity center	Activity	Description
1. Reuse infrastructure creation	1. Develop reuse infrastructure	Develop reuse policies, tools, processes, measures
2. Reuse infrastructure mainte-nance and reuse marketing	2. Maintain reuse infrastructure	Maintain reuse policies, tools, processes, measures
	3. Communicate existence of components	Make potential reusers aware of the existence of reusable components
3. Reuse administration	4. Administer reuse measurement, accounting and incentives	Measure ongoing levels of reuse on projects. Allocate costs of reuse. Reward in-dividuals and/or teams for reuse achieved
4. Reuse production	5. Analyze reuse opportunities	Investigate opportunities for acquiring or develop-ing components
	6. Develop or acquire reusable components	Develop components to be reusable; acquire and/or generalize components previously developed
	7. Certify components	Certify components offered for reusability
	8. Document, classify and store components	Document, classify and store offered components
5. Reuse consumption	9. Search for components	Search libraries for com-ponents (or assist the search process)
	10. Retrieve, understand and evaluate components	Understand and evaluate components found in libraries
	11. Adapt and integrate components	Modify (if necessary) and integrate components
6. Reuse maintenance	12. Maintain reusable components	Correct and extend compo-nents in the library
	13. Update reusable components	Correct and extend compo-nents in systems

[a] Goldberg and Rubin [14], with adaptation.

braries. We base this assumption on the observation that at ComputerCo, like most soft-ware organizations, the primary focus of managerial attention from a costing standpoint is on the software development project. Other possible cost objects include departments, products, or product families. An alternative chart of accounts would be appropriate if one of these alternative foci were chosen.

Our fourth assumption is that the target organization wishes to assign *all* reuse-related costs to projects, including those that are more remote from projects themselves. These include activities #1 (Develop Reuse Infrastructure), #2 (Maintain Reuse Infrastructure), and #3 (Communicate Existence of Components). For these activities we propose using the standard cost of the activity per unit of reuse consumed as the basis for allocating costs.[3] An example provides the rationale for this approach. Suppose that an organization has a total of 1000 units of reuse (e.g., components weighted by size/complexity) by various project teams in the analysis period used to determine standard values for the ABC model. Further, suppose the cost of Infrastructure Maintenance (Activity #2) for that period is $500,000, making the standard cost of Infrastructure Maintenance per unit of reuse $500. This cost is then used as the basis for allocating costs to projects going forward; e.g., a project that had 10 units of reuse consumed would be allocated a cost for Infrastructure Maintenance of $5000. We believe that this approach provides a reasonable basis for allocating these costs to projects, and certainly provides a significant improvement in accuracy over conventional approaches (e.g., allocating costs based on project size without regard to the level of reuse consumed).

Our fifth assumption is that the organization views the production of reusable components not as an end in itself, but rather, as an enabler of subsequent reuse consumption. Under this view, a project that produces reusable components is just the first of several projects consuming those components. The implication of this assumption is that the costs associated with reuse production (activities #5–#8) should be allocated to projects based on the extent of reuse *consumption* rather than on the extent of reuse *production*. To implement this approach, we use the same procedure as for the activities #1–#3 described above, i.e., we base cost allocation on the standard cost of the activity per unit of reuse consumption.

Example activity: develop reuse infrastructure

A full presentation of all thirteen activities is beyond the format restrictions of the current article, but is available from the authors.[4] As an example, presented below is the first of the thirteen identified activities, "Develop Reuse Infrastructure", which encompasses all work involved in setting up reuse policies, tools, processes, measures, initial reusable parts, training materials etc. required to support the process of reuse in the organization. This activity only covers the one-time effort to set up the infrastructure for an organization. On-going maintenance is viewed as a separate activity. The ABC model for this activity is summarized in figure 2 and defined in detail in table 4. To iden-

---

[3] In most economic models a unit of reuse is defined as a component that can be used as-is to perform some well defined function [11]. We expect that it may be advisable to weight components by size, complexity, or standard development cost (e.g., [18]) when doing cost allocation, although there are important tradeoffs to be considered between the accuracy of costing and the cost of implementing the costing program [17, p. 104].

[4] The full set of thirteen figures and thirteen tables is currently available (2000) on the WWW at: http://www.pitt.edu/~ckemerer/kemerer.htm.
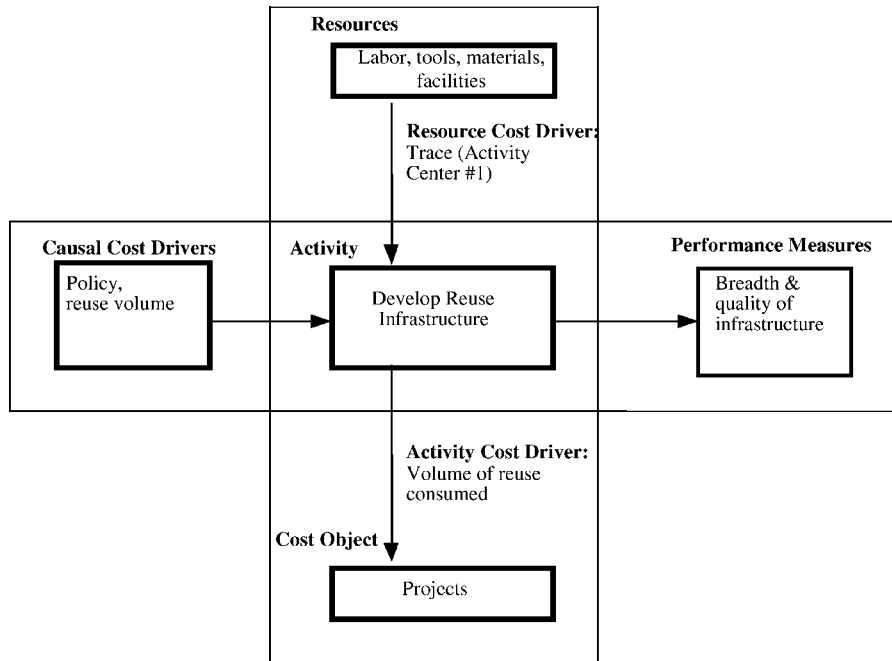
Figure 2. ABC chart for "Develop Reuse Infrastructure" activity.

tify candidate resource cost drivers, activity cost drivers, and causal cost drivers (for this and the other twelve activities) we followed the procedure described in Turney [27]. To support this effort, we also drew upon knowledge of the nature of reuse activities gained from observations at ComputerCo, and a review of the literature on reuse.

In this proposed environment attention is focused on activities devoted to providing for the infrastructure necessary for component reuse to take place. As can be seen from table 4, a variety of performance measures are possible. The expectation is that organizations would initially use relatively simple and/or easy to collect measures, and then refine these with experience. Eventually, it would be expected that these measures would become automated in a mature process [1,2].

Summaries of the ABC model from the cost allocation view and the process view are provided in tables 5 and 6 below, respectively. Through use of a management model such as this one, it is expected that organizations embarking on a component-based software engineering strategy for their products will be able to make more rational decisions about the true costs and benefits of such an approach, and will allow for appropriate incentives to be created to encourage software reuse to its fullest economic extent.

6.   Summary and conclusions

For a large-scale component reuse environment to thrive, an organization must devote considerable resources to reuse activities (e.g., maintaining reuse infrastructures, devel-

Table 4
ABC chart for "Develop Reuse Infrastructure" activity.

ABC element	Short description	Detailed description
Activity	Develop reuse infrastructure	Design and implement reuse policies, tools, processes, measures, etc.
Resources	Labor, tools, materials	Labor, tools and materials (e.g., acquired parts) employed to create the reuse infrastructure.
Resource cost driver	Trace to Activity Center 1	Tracing procedure: Aggregate all costs for infrastructure development (Activity Center 1) and allocate them to the period in question (e.g., via amortization). Divide the allocated cost by the total volume of components reused by all projects in the period to get the standard unit cost of infrastructure development.
Activity cost driver	Volume of reuse consumed by a project	The number of components (weighted by size/complexity) reused by a project. This number times the unit cost (see "Resource Cost Driver") determines the cost allocation to the project for this activity.
Cost object	Project	A project that reuses components from shared libraries.
Overall cost driver	Policy, expected reuse volume	The reuse policy determines how aggressively reuse will be sought. The expected reuse volume is the total expected extent of reuse for the planning horizon (e.g., number of component reuses, target project reuse percentage). A more aggressive reuse policy and a greater expected reuse volume justify a greater investment in reuse infrastructure.
Performance measures	Breadth and quality of reuse infrastructure	Size of reuse libraries: number of components (weighted). Quality of reuse libraries: average certification level, average frequency of reuse. Scope of reuse tools: rated coverage of reuse tools (e.g., for searching, retrieving) against an industry standard. Quality of reuse tools: rated quality of reuse tools against some industry standard.

oping and maintaining components, locating and evaluating components, adapting and integrating components) that will be performed by some project teams on behalf of others, or, more likely, will be performed by reuse specialists residing entirely outside of traditional project teams. Yet, while such a reorganization of work is required to achieve the benefits of component-based development, it also causes potentially dramatic increases in the proportion of total project costs attributable to indirect and overhead costs associated with reuse, and thus demands some rethinking of traditional approaches to project cost allocation and cost management. As we have explained in this article, ABC provides a way to address both of these issues. On the cost allocation side, ABC allows a more accurate mapping between the indirect and overhead costs associated with reuse on the one hand, and the projects that actually consume reuse activities on the other. On

Table 5
Component-based reuse ABC cost allocation view summary.

Activity	Resources	Resource cost driver	Basis for calculating unit costs	Activity cost driver (cost object is the project)
1. Develop reuse infrastructure	Labor, tools, materials, facilities	Trace to activity center #1	Standard cost per reused component per period	Volume of reuse consumed on a project
2. Maintain reuse infrastructure	Labor, tools, materials, facilities	Trace to activity center #2	Standard cost per reused component per period	Volume of reuse consumed on a project
3. Communicate existence of components	Labor, tools, materials, facilities	Trace to activity center #2	Standard cost per reused component per period	Volume of reuse consumed on a project
4. Administer reuse measurement, accounting and incentives	Labor, tools, materials, facilities	Trace to activity center #3	Standard cost per reuse measure	# measurements for project
5. Analyze reuse opportunities	Labor, tools, materials, facilities	Trace to activity center #4	Standard cost per reused component per period	Volume of reuse consumed on a project
6. Develop or acquire reusable components	Labor, tools, materials, facilities	Trace to activity center #4	Standard cost per reused component per period	Volume of reuse consumed on a project
7. Certify components	Labor, tools, materials, facilities	Trace to activity center #4	Standard cost per reused component per period	Volume of reuse consumed on a project
8. Document, classify and store components	Labor, tools, materials, facilities	Trace to activity center #4	Standard cost per reused component per period	Volume of reuse consumed on a project
9. Search for components	Labor, tools, materials, facilities	Trace to activity center #5	Standard cost per search	# searches for project
10. Retrieve, understand and evaluate components	Labor, tools, materials, facilities	Trace to activity center #5	Standard cost per evaluation	# evaluations for project
11. Adapt and integrate components	Labor, tools, materials, facilities	Trace to activity center #5	Standard cost per integration	# integrations for project
12. Maintain reusable components	Labor, tools, materials, facilities	Trace to activity center #6	Standard cost for component maintenance	# maintained library components for project
13. Update reusable components	Labor, tools, materials, facilities	Trace to activity center #6	Standard cost for component maintenance	# maintained system components for project

Table 6
Reuse ABC process view summary.

Activity	Causal cost driver	Performance measure
1. Develop reuse infrastructure	Reuse policy Expected reuse volume	Breadth and quality of infrastructure
2. Maintain reuse infrastructure	Size/complexity of infrastructure Volume of reuse	Breadth and quality of infrastructure
3. Communicate existence of components	Size of reuse libraries Size of developer community	Number of communications
4. Administer reuse measurement, accounting and incentives	Reuse policy Volume of reuse	Number of measurements
5. Analyze reuse opportunities	Reuse policy	Number of components evaluated
6. Develop or acquire reusable components	Reuse opportunity	Number and quality of reusable components acquired
7. Certify components	Offered components	Number of components certified
8. Document, classify and store components	Certified components	Number of components stored
9. Search for components	Project policy Reuse potential of the domain Size/quality of reuse libraries	Number of searches Number of promising components found
10. Retrieve, understand and evaluate components	Promising components found	Number components evaluated, number of components accepted
11. Adapt and integrate components	Retrieved components selected for use	Number of attempted integrations, number of successful integrations
12. Maintain reusable components	Volume of reuse	Number of components maintained
13. Update reusable components	Volume of reuse	Number of components updated

the cost management side, ABC can support organizational efforts to understand what drives the costs associated with reuse activities, and how to improve the efficiency and effectiveness of these activities.

The ABC model we have developed here represents the first step in bringing the potential benefits of ABC to organizations adopting large-scale component reuse. But there is much more work to be done. As more organizations move from a custom software development approach to one that more closely resembles a manufacturing process, opportunities will arise for researchers and practitioners to design and evaluate ABC programs along the lines outlined here. Just as it has taken time for researchers and practitioners

to determine the most effect approach to administering ABC in a manufacturing environment, we expect that much empirical work will be needed to establish which activity breakdowns apply best in software development environments, to establish reasonable bounds on the standard costs associated with various resource and activity cost drivers, and to gauge how much effort should go into administration of the ABC program itself. Other interesting work could examine the effect of the adoption of an effective ABC program on incentives for reuse and actual levels of software reuse achieved. A final avenue of research could examine the effect of ABC on the cost effectiveness of component-based software development itself.

Acknowledgements

Appendix A. Work breakdown structure details

A.1. Systems development and maintenance

1. FUNCTIONAL REQUIREMENTS ANALYSIS
   Handling of external requests for products and changes to existing products.
   Work with specific competitive products either in the lab or on paper to understand competitive offerings and build competitive products.

2. PLAN AND PROJECT MANAGEMENT
   Operational project management which coordinates and controls day-to-day software development activities.

3. PRODUCT DESIGN
   Architecture
   Definition of technical strategies for the environment that the program is to be a part of, i.e., the set of rules that the program must follow to be a part of a larger system.
   Standards – formal and informal.
   Detailed definitions of what the program will do and how it will appear to its users.
   Includes formal design reviews.
   Usability planning and design
   Efforts done during the development cycle to "improve" the ease of use of a product.
   Task analysis, competitive evaluation (by customers), walk-throughs of proposed design, prototype development and evaluation.

4. INTERNAL DESIGN

   Detailed definitions of how the program will operate, i.e., the components that the program will be made up of the role/responsibility of each component, and the relationship of the components (e.g., the flow).

5. REUSE PLANNING AND ACQUISITION

   Planning, design, location, and acquisition of designs, code, test suites, and/or documentation, etc. for reuse.

   Planning, design, location, and acquisition of designs, codes, test suites, and/or documentation, etc. for 00 implementation.

   Training to develop new skills for reuse or 00.

   Any other re-use or 00 activity.

6. CODE AND UNIT TEST

   Writing of the programming instructions necessary to execute the program.

   Segments/units/modules of the code are then tested to ensure that they perform according to specification.

   Includes formal design reviews.

7. OTHER PROGRAMMING SUPPORT

   Tool development

   Creation and testing of tools to support development and to improve development productivity unique to a specific product or set of products, e.g., simulators, design tools.

   Includes acquisition of tools and equipment to perform the testing.

   National Language Support (*NLS*) *translation*

   Includes the cost of product enablement.

   Includes the code of programming for translation.

8. TEST PLANNING

   This includes all activities related to planning for testing of all or part of a product to determine that it does what it is intended to do. See Verification and Validation below for details.

   - Product verification and systems test.
   - Performance analysis and measurement.
   - Customer test.

9. COMPONENT TEST AND PRODUCT VERIFICATION TEST

   Verification and planning that individual components execute as they are intended to execute.

   Verification that the product can perform the individual functions that it is intended to perform, in a controlled environment, with all of the necessary components operating together.

10. SYSTEMS TEST
    Verification that components can work together as intended.
    Verification that the product can be installed as directed.
    Verification that the product will operate as advertised when executed as a part of a larger system in a somewhat uncontrolled environment.
    Verification that the product will operate properly when placed in simulated customer environments.

11. PERFORMANCE ANALYSIS AND MEASUREMENT
    Analysis of the performance characteristics of the system, design guidance for performance improvements, debugging of performance problems, system positioning, and development of internal performance tools, measurement of the product.

12. OTHER VERIFICATION AND VALIDATION ACTIVITIES
    Usability verification
    Efforts in the development cycle to evaluate or measure the usability of a product. Unique activities that focus on the time to perform a task or the accuracy with which a user can perform a task.
    Customer test
    Testing of code shipped earlier than the version intended for general availability such as beta test or early customer validation.

13. CONFIGURATION MANAGEMENT AND QUALITY ASSURANCE
    Bringing together all of the individual segments/units/modules to create the components of the product so that they can be put together for execution.

14. INFORMATION DEVELOPMENT
    This includes all activities that help the users (or potential users) understand the product. The results of these activities may be included in the products themselves (such as "help") and it may include materials that complement the products (such as documentation).

15. MAINTENANCE
    This includes all activities for direct support of customers when they have problems using the product (i.e., they have purchased the product). This includes, but is not limited to the following:

    - on-site local support;

    - answering questions of how the product works;

    - problem determination to decide if the user has encountered a program defect.

16. MAINTENANCE
    Fix correction
    Correction of program defects (bugs) as well as verification and distribution of the fixes and control of fix libraries and documentation.

17. MAINTENANCE

Service and support coordination
Education, training, skills building (includes skills transfer from development to service personnel).
Tools development.
Quality and Performance management.
Contract management.

18. MAINTAINING DEVELOPER WORKSTATIONS

These activities associated with establishing and maintaining a distributed or workstation environment for development and/or e-mail purposes. Included is the time/expenses associated with HW/SW configuration, update, backup and recovery, maintenance, and connectivity.

A.2. Program products management and planning

19. PROJECT OFFICE FUNCTIONS

Announce planning
Preparing all the plans and materials to announce the product or changes to a product.
Contract management
Plan and manage agreements (contracts) with suppliers and customers.
OEM relations
Major responsibilities for establishing and/or managing external alliance/vendor/OEM activities.
National language support
This is made up of those activities necessary for interfacing to translation centers for changing screen printed output from one national language to another. It does not include changes from one programming language to another. Also, it does not include programming activities necessary to make it possible to do translation.
This activity also includes translation planning which is the definition of what translation is to be done – e.g., identification of the languages that screens are to be translated into.

20. PRODUCT AND MARKET PLANNING

Strategy development
Analysis of marketplace to set a long-term direction for new and enhanced products.
Product/market planning
Those activities that relate to understanding the market for a product.
This includes market segmentation, opportunity analysis, market research and evaluation (including competition).

A.3. Program supervision and general

21. COMMON GENERAL DEVELOPMENT ENVIRONMENT/TOOLS AND SUP-
PORT
Software environment support
Includes those things other than "IS support" and "tool development" that provide
things necessary and helpful in development of products. For example, this might
include development of very general tools and new programmer training.
Process evaluation not related to a single product.
Non-direct staff; i.e., secretaries and administrative.
Non-direct management; i.e., upper management not directly charged to a product.
Quality management
Activities such as planning, monitoring, and reporting product quality against goals
and activities for continuous product quality improvement. Includes normal day to
day project activities.

22. MANAGEMENT SUPERVISION
People management, development, performance, and salary administration.
Activities that support the people of the organization and their needs; e.g., benefits,
space planning, general communications.

23. ADMINISTRATIVE SUPPORT
Major responsibilities for establishing and tracking of budgets.
Meetings of a general nature both formal and informal.
Creating, contributing to, or making reports and presentations not associated with
designing, building, or testing SW.

A.4. Sales, general and administrative (SG&A) activities

24. MARKETING SUPPORT
This is made up of all activities that are in support of a marketing and sales organi-
zation. This includes, but is not limited to, the following:

- education and training for internal and external customers;
- provide technical support to the sales organization;
- customer relationship management;
- supporting customers with early code.

25. DISTRIBUTION
Includes all preparation of images and packages for external shipment of the prod-
uct.
This also includes all activities needed to create and management Distribution and
Support plans.

26. MARKETING COMMUNICATION
   Plan and manage all marketing communications.
   Producing marketing literature.
   Trade shows and User groups.

27. MARKETING DEVELOPMENT
   Functional (people expense) support of market-focused programs such as industry, channel, installed base program or market segment support and development (mainframe alternative, client/server, customer contact, etc.)
   Indirect channel partner business development, account management and recruiting funds (excluding porting funds, which is counted as an R&D).
   Management and infrastructure of market development programs, like industry, indirect channel, installed base, direct marketing and competitive programs. Global/major account marketing activities not included in field selling costs.

References

[1] R.D. Banker, R.J. Kauffman, C. Wright and D. Zweig, Automating output size and reuse metrics in a repository-based computer-aided software engineering (CASE) environment, IEEE Transactions on Software Engineering 20 (1994).

[2] J.M. Bieman and S. Karunanithi, Measurement of language-supported reuse in object-oriented and object-based software, Journal of Systems and Software 30 (1995) 271–293.

[3] G. Caldiera and V.R. Basili, Identifying and qualifying reusable software components, IEEE Computer 24 (1991) 61–70.

[4] G. Cokins, *Activity Based Cost Management: Making it Work* (McGraw Hill, 1996).

[5] B. J. Cox, Planning the software industrial revolution, IEEE Software 7 (1990) 25–33.

[6] M. A. Cusumano, Shifting economies: From craft production to flexible systems and software factories, Research Policy 21 (1992) 453–480.

[7] G. DeSanctis and M.S. Poole, Capturing the complexity in advanced technology use: Adaptive structuration theory, Organization Science 5 (1994) 121–147.

[8] T.L. Estrin, Is ABC suitable for your company? Management Accounting 75 (1994) 40–45.

[9] D. Fafchamps, Organizational factors and reuse, IEEE Software 11 (1994) 31–41.

[10] R.G. Fichman and C.F. Kemerer, Object technology and reuse: Lessons from early adopters, IEEE Computer 30 (1997) 47–59.

[11] W. Frakes and C. Terry, Software reuse: Metrics and models, ACM Computing Surveys 28 (1996) 425–435.

[12] W.B. Frakes and S. Isoda, Success factors of systematic reuse 11 (1994) 14–19.

[13] J.E. Gaffney Jr. and T.A. Durek, Software reuse – key to enhanced productivity: some quantitative models, Information and Software Technology 31 (1989) 258–267.

[14] A. Goldberg and K. Rubin, S, Reuse process models, in: *Succeeding with Objects (Decision Frameworks for Project Management)*, eds. A. Goldberg and K. Rubin (Addison-Wesley, 1995).

[15] M.L. Griss, Software reuse: From library to factory, IBM Systems Journal 32 (1993) 548–566.

[16] R.S. Kaplan, *Measures for Manufacturing Excellence* (HBS Press, 1990).

[17] R.S. Kaplan and R. Cooper, *Cost & Effect* (HBS Press, 1998).

[18] C.F. Kemerer, Reliability of function points measurement: A field experiment, Communications of the ACM 36 (1993) 85–97.

[19] Y. Kim and E.A. Stohr, Software reuse: Survey and research directions, Journal of Management Information Systems 14 (1998) 113–147.

[20] C.W. Krueger, Software reuse, ACM Computing Surveys 24 (1992) 131–183.

[21] B. Meyer, Reusability: The case for object-oriented design, IEEE Software (1987) 50–64.

[22] H. Mili, F. Mili and A. Mili, Reusing software: Issues and research directions, IEEE Transactions on Software Engineering 21 (1995) 528–562.

[23] J.A. Ness and T.G. Cucuzza, Tapping the full potential of ABC, Harvard Business Review 73 (1995) 130–138.

[24] S.L. Pfleeger and T.B. Bollinger, The economics of reuse: New approaches to modeling and assessing cost, Information and Software Technology 36 (1994) 475–484.

[25] J.S. Poulin, J. Caruso and D. Hancock, The business case for software reuse, IBM Systems Journal 32 (1993) 567–594.

[26] J.S. Poulin, Populating software repositories: Incentives and domain-specific software, Journal of Systems Software 30 (1995) 187–199.

[27] P.B. Turney, *Common Cents*, Cost Technology (1991).

[28] M. Wasmund, Implementing Critical Success Factors in software reuse, IBM Systems Journal 32 (1993) 595–611.