Record: 1	
Title:	Reusable asset. (cover story)
Authors:	Kemerer, Chris F.
Source:	InformationWeek; 09/22/97 Issue 649, p64, 3p, 1c
Document Type:	Article
Subject Terms:	COMPUTER software -- Development
	NAICS/Industry Codes51121 Software Publishers
Abstract:	Focuses on the proposed technical solutions to the costs and difficulties to improving software development. Weaknesses of software development approach; Activity based-costing for developing product costs; Levels of reusable components; Software organizations that implement reuse. INSETS: Keys to reuse; Inhibitors to reuse.
Full Text Word Count:	1599
ISSN:	8750-6874
Accession Number:	9710115173
Persistent link to this record:	http://search.epnet.com/login.aspx?direct=true&db=buh&an=9710115173
Cut and Paste:	<A href="http://search.epnet.com/login.aspx?direct=true&db=buh&an=9710115173">Reusable asset.</A>
Database:	Business Source Premier

Section: 500: SOFTWARE

REUSABLE ASSET

Object-oriented tools and iterative development methods can save companies money and satisfy clients

Much attention has been devoted in recent years to the cost and benefits of computer hardware, with numerous articles, reports, and seminars on, for example, migrating an organization to client-server architectures. But given the fact that U.S. organizations spend an estimated $20 billion each year on custom software alone, it's surprising that significantly more analysis has not been devoted to improving the return on software development.

Up to now, most work has focused on proposed technical solutions to the costs and difficulties of developing software, or lamented the need to spend half or more of all resources on "maintaining" existing software. What's needed is a new way of thinking about software, one that treats it as a valuable asset.

Meanwhile, the nature of software development is undergoing a dramatic change, one that strikes at the core of how major corporate users of information technology will develop software over the coming years. Traditionally, large systems have been custom-built, using a "waterfall" - style approach to development. Under this approach, a project begins with a requirements or needs analysis, then proceeds through stages of design, coding, testing, documentation, training, and implementation.

This custom-crafted approach to software development has long had two major weaknesses. First, such an approach 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-including not only program code, but also analysis and design artifacts-created for previous systems.

While this issue has been widely recognized, until recently modern software development technologies made reuse possible on only an ad hoc or local basis. Large-scale reuse was unusual.

Second, such an approach to large systems delays the primary acceptance by users until the very end of the project, when requested changes are the most expensive. In addition, the sheer complexity of these systems and the limited client-developer interaction and communication during the early phases of the project make design errors, and therefore required changes, likely.

In response to these two primary issues, a new approach has been proposed for The development of these systems. Using object-oriented development tools and iterative development methods, many observers have suggested what amounts to a manufacturing approach to software development.

This enhanced delivery capability should simultaneously result in cost savings, as the object-oriented approach allows for the reuse of high-quality components, and a higher level of client satisfaction, as systems are developed iteratively with much client interaction and feedback.

However, these optimistic scenarios contain several management challenges in terms of both product and process issues. On the product side, the development of reusable components for later assembly into various systems is a radical departure from the current, single-use, project-based approach that dominates systems-development activities.

One of the reasons that only limited reuse is done today is that project managers have limited incentives for such activity. In fact, they often have strong disincentives, in terms of the likely negative impact on project budget and schedules that activities such as investing in reusability often entail.

An approach that emphasizes the development of reusable components will Require strong incentives for component development, perhaps even to the point of creating a separate organization focused on the delivery of reusable components.

A related issue concerns how such items should be costed. 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.

From an organizational perspective, given expected reuse, the software development team need not receive full cost recovery on a component's first use. Also, a team assembling an application will need to carefully consider its own costs to develop a component if a similar one is already available at a reduced cost.

Coupled with these product changes are the process changes in moving to an iterative development strategy. Managers have developed a whole series of Checklists and other guides about how best to manage linear-style custom software development projects. Essentially, all of these guides and checkpoints will become obsolete under the new iterative development approach.

For organizations that wish to move to this new approach to software development, it makes sense to first understand how similar processes in manufacturing are managed.

True Costs

The most effective technique is known as activity-based costing (ABC). This is an approach originally designed as a way to allocate overhead in manufacturing settings in order to develop true product costs. Prior methods of product costing typically relie on assumptions about the manner in which products were developed years ago--assumptions that changed with increasing automation and which today can produce startling anomalies in actual costs vs. standard costs.

These anomalies can lead to poor management decision-making with regard to product pricing, customer marketing and service, and facilities and environment acquisition and retention. From its original base in product costing, the ideas behind ABC have been extended and generalized to what has activity-based management (ABM). This is a process-based focus on managing for more effective results, and is similar in intent to some of the work done as "reengineering. " The basic idea is to identify those activities or tasks that consume resources and to focus process-improvement actions on the high-leverage activities.

Even from these brief descriptions, it should be clear how these two ideas relate. A careful study of the process in which critical activities are identified can, in addition to being used to improve the process, also significantly influence a decision to cost the products that results from such processes.

In addition, the management of software development will require a new way of accounting for software costs in order to move from the use-it-once, project-based mode of development to a reusable components style.

The reuse process at most organizations today is opportunistic rather than systematic. With few exceptions, reuse is not measured, and there are few guidelines, processes, incentives, or tools in place to encourage or enforce reuse per se.

This doesn't mean that reuse is not happening. Many sites, including a number of companies in the InformationWeek 500, report at least some reuse, and a few report extensive reuse. What it does mean is that reuse tends to be driven by needs and opportunities as they rise during the course of the projects hand.

Many Faces Of Reuse

Many levels of reuse are possible, typically ranging through the following: (1) retrofitting an existing product to create a new product; (2) porting an existing product to a new environment; (3) factoring common code so it exists in only one place within a product; (4) subclassing within an object-oriented product; (5) subclassing from a class library developed elsewhere; (6) developing a generator that creates specific instances from a template; and (7) reusing services from a centrally managed library.

Over several years of research with various colleagues, including Michael Cusumano at MIT and Rob Fichman at Boston College, I have studied a number of software organizations' attempts to implement reuse.

While the majority have lacked a systematic reuse process, a few leading organizations are doing substantial amounts of reuse. In today's environment we observed some common elements that tended to be present at these successful sites, but many opportunities for reuse are missed due to a variety of inhibitors (see graphics, p. 65).

In a systematic reuse environment, indirect costs and overhead associated with reuse can become a significant portion of software-development costs, possibly approaching 50%.

Traditional methods of tallying indirect costs and overhead (for example, based on direct labor hours or project costs) are not an effective means of allocating costs in such an environment. 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.

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

While the specific activities involved in software reuse are beyond the scope of this article and will likely vary across organizations, they can be expected to include such categories as reuse production and consumption, reuse maintenance, and reuse infrastructure development.

Management needs to put into place a structure and tools that appropriately encourage a reuse orientation to software development. Companies need to think of existing software as an asset that is available for reuse, and see new projects as opportunities for such reuse. Activity-based costing provides a basis for moving toward these goals.

Keys to Reuse

Strong edict to seek reuse

Limited resources for reinvention

Good-quality, directly applicable assets were available that Could not be ignored

Very small core development team (four to eight people)

Well-understood, narrowly defined functional requirements

Strong central architect who acts as reuse guardian

Absence of special performance requirements

Inhibitors To Reuse

Reuse decisions made by individual developers

Only measurements are project schedule and budget

Object-oriented components often must be rewritten before they are done right

Assets not known to meet operational or performance needs of new project

Insufficient incentives for reuse across team boundaries

Ownership and version management issues for reuse across teams not addressed

~~~~~~~~

By Chris F. Kemerer

Chris F. Kemerer, Ph.D., is the David M. Roderick Chair in Information Systems at the Katz Graduate School of Business, University of Pittsburgh. Previously, he was an associate professor at MIT's Sloan School of Management. He can be contacted at ckemerer @katz.business.pitt.edu, and his home page is www.pitt.edu/~business/faculty/kemerer.htm.

Top of Page

Back
~~~~~~~~