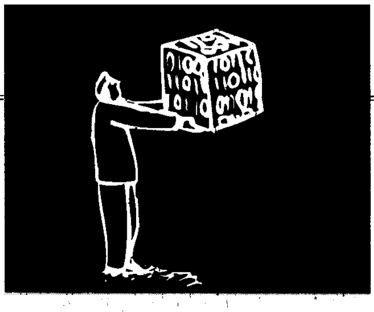


Effects of Reuse on Quality, Productivity, and Economics

WAYNE C. LIM

◆ *Metrics collected in two reuse programs at Hewlett-Packard demonstrate improved quality, increased productivity, and reduced time to market. The results of economic cost-benefit analyses indicate reuse can provide a substantial return on investment.*

Although not a new concept, reuse as a means of improving software quality and productivity has been aggressively pursued only recently. Hewlett-Packard has found that reuse can have a significant and largely positive effect on software development. This article presents metrics from two HP reuse programs that document the improved quality, increased productivity, shortened time-to-market, and enhanced economics resulting from reuse.

In this article, *work products* are the products or by-products of the software-development process: for example, code, design, and test plans. *Reuse* is the use of these work products without modification in the development of other software. *Leveraged reuse* is modifying existing work products to meet specific system requirements. A *producer* is a creator of reusable work products, and the *consumer* is someone who uses them to create other software. *Time to market* is the time it takes to deliver a product from the time it is conceived.

Our experience with reuse, which includes multiple reuse programs in different divisions within the company, has been largely positive. Because work products are used multiple times, the accumulated defect fixes result in a higher quality work product. Because the work products have already been created, tested, and documented, productivity increases because consumers of reusable work products need to do less work. However, increased productivity from reuse does not necessarily shorten time-to-market. To reduce

0740-7459/94/$04.00 © 1994 IEEE

TABLE 1
QUALITY, PRODUCTIVITY, AND TIME-TO-MARKET PROFILES

Organization	Manufacturing Productivity	San Diego Technical Graphics
Quality	51% defect reduction	24% defect reduction
Productivity	57% increase	40% increase
Time to market	Data not available	42% reduction

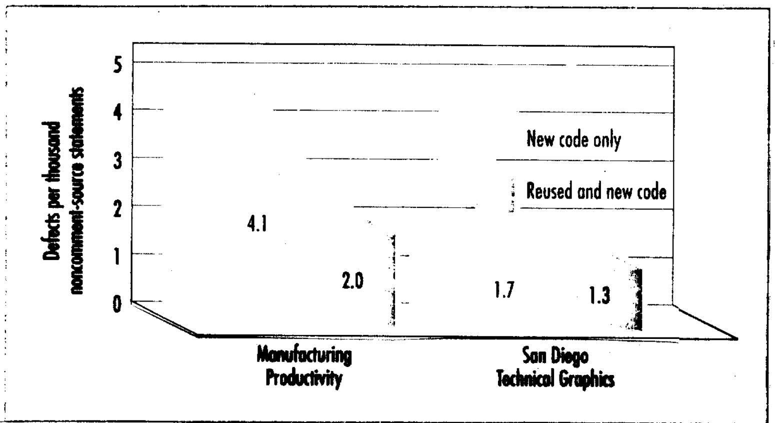


Figure 1. Effect of reuse on software quality — as measured by defects per thousand noncomment source statements — in new code only versus new code combined with reused code, in two development efforts participating in the two HP reuse programs analyzed.

time-to-market, reuse must be used effectively *on the critical path* of a development project, the chain of activities that determine the total project duration. Finally, we have found that reuse allows an organization to use personnel more effectively because it leverages expertise. Experienced software specialists can concentrate on creating work products that less experienced personnel can then reuse.

However, software reuse is not free. It requires resources to create and maintain reusable work products, a reuse library (if necessary), and reuse tools. To help evaluate the costs and benefits of reuse, we have developed an economic analysis method, which we have applied to multiple reuse programs at HP. I present the results from two of these programs here.

TWO CASE STUDIES

The first reuse program is within the Manufacturing Productivity section of HP's Software Technology Division, which produces large-application software for manufacturing resource planning. The MP section's reuse program started in 1983 and is ongoing. The original motivation for pursuing reuse was to increase engineering productivity to meet critical milestones.[1] The MP section has since discovered that reuse also eases the maintenance burden and supports product enhancement.

MP engineers practiced reuse by using generated code and other work products such as application and architecture utilities and files. The data reported here reflects only the use of reusable work products, not generated code. Total code size for the 685 reusable work products was 55,000 lines of noncomment source statements. The reusable work products were written in Pascal and SPL, the Systems Programming Language for the HP 3000 computer system. The development and target operating system was MPEXL, the Multiprogramming Environment.

The second program is within the San Diego Technical Graphics Division, which develops, enhances, and maintains firmware for plotters and printers. The STG reuse program began in 1987 and continues to the present. Among the program's goals, as described in an internal report, were lowering development costs by reducing duplication and providing consistent functionality across products. The reusable work product analyzed here is 20,000 noncomment source statements written in C. The development operating system was HPUX and the target operating systems were PSOS and an internal one.

FINDINGS

At HP, we collected data from these two reuse programs and conducted a *reuse assessment* — an analytical and diagnostic method used to evaluate both qualitative and quantitative aspects of a reuse program. As part of this assessment, data on the improved quality, productivity, and economics attributable to reuse is analyzed and documented. Table 1 summarizes the productivity, quality, and time-to-market benefits from reuse.

Quality. Because work products are used multiple times, the defect fixes from each reuse accumulate, resulting in higher quality. More important, reuse provides incentives to prevent and remove defects earlier in the life cycle because the cost of prevention and debugging can be amortized over a greater number of uses.[2]

Figure 1 summarizes the quality results. The MP section's data shows a defect-density rate for reused code of about 0.9 defects per thousand noncomment source statements (KNCSS) (not shown in figure) compared to 4.1 defects/KNCSS for new code. Using reused code in combination with new code (in which 68 percent of the product was from reused work products) resulted in 2.0 defects/KNCSS, a 51 percent reduction in defect density compared to new code. If we take into account the effects of generated code, we achieve a total defect-density reduction of 76 percent compared to new code.

The STG division also reported a positive experience with reuse. They estimated the actual defect-density rate for reused code to be 0.4 defects/KNCSS (not shown in figure), compared to 1.7 defects/KNCSS for new code. A product that incorporated the STG reusable work product had a 31 percent reuse level and a defect-density rate of about 1.3 defects/KNCSS, a 24 percent reduction in defect density.

Productivity. Reuse improves productivity because the life cycle now requires less input to obtain the same output. For example, reuse can reduce labor costs by encouraging specialization in areas such as user interfaces. Because of their experience, specialists usually accomplish tasks more efficiently than nonspecialists. Or productivity may increase simply because fewer work products are created from scratch. For example, if the reused work products are already documented and tested, the new product requires less work in these areas. Reuse can also improve a product's maintainability and reliability, thereby reducing maintenance labor costs.

In general, reuse improves productivity by reducing the amount of time and labor needed to develop and maintain a software product. As Figure 2 shows, another similar project in the MP section reported a productivity rate of 0.7 KNCSS/engineering month for new code. Its product, which was composed of 38 percent reused code, had a productivity rate of 1.1 KNCSS/engineering month, a 57 percent increase in productivity over development from scratch.

The STG division estimates a productivity rate of 0.5 KNCSS/engineering month for new code. In contrast, its released product comprising 31 percent reused code had a productivity rate of 0.7 KNCSS/engineering month, a 40 percent improvement.

Another firmware division within HP has been tracking the reuse ratio to the productivity rates in the development of their products. As Figure 3 shows, by 1987 several products had

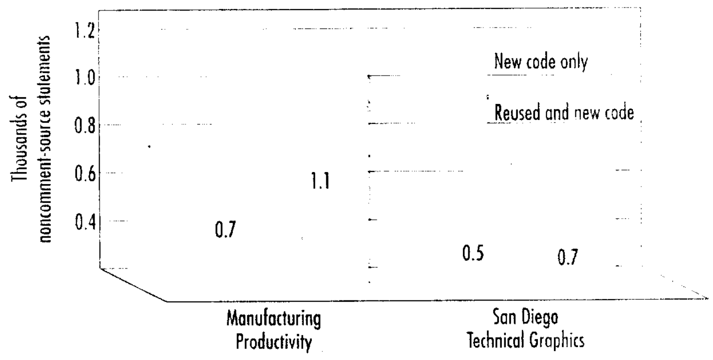


Figure 2. Effect of reuse on productivity — as measured by thousands of noncomment source statements produced per engineering month — in new code only versus new code combined with reused code, in two development efforts participating in the two HP reuse programs analyzed.

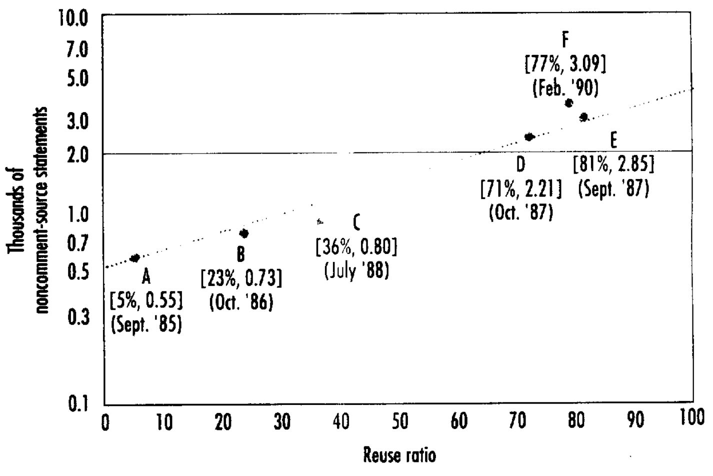
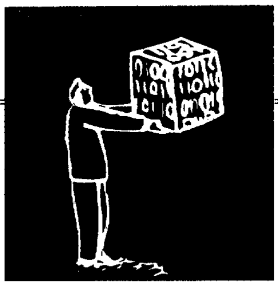


Figure 3. Firmware productivity in a third HP division, measured from 1985 through 1990. Using reuse, the division had set a productivity goal of two thousand noncomment source statements per engineering month by 1990 (dotted line). The division exceeded its goal with the release of product D in 1987. The numbers in brackets associated with each product are the percentage of reused code in the product and the thousands of noncomment source statements actually produced each engineering month. Note that the ratio used here includes leveraged code, which is reused code that has been modified.



already exceeded their 1990 productivity goal of 2.0 KNCSS/engineering month with greater than 70 percent reuse. And they were well above their projected productivity rates. It should be noted that the reuse ratio calculation used in this division,

reuse ratio = ([directly reused KNCSS + modified KNCSS]/ product total KNCSS) x 100

includes leveraged reuse of code as well.

Time to market. The STG division reports that the same development effort using the reusable work product required only 21 calendar months compared to an estimated 36 calendar months had the reusable work product not been used, a reduction of 42 percent. Suitable data showing elapsed time was not available for the MP product.

REUSE COSTS

In general, the costs of reuse include creating or purchasing reuse work products, libraries, tools, and implementing reuse-related processes. I will not explicitly describe them here, but will include them in aggregate form in the next section. In this section, I focus on the incremental cost to create a reusable work product.

Techniques to create reusable work products range from reengineering existing work products to be reusable to intentionally developing new work products for reuse. Table 2 summarizes some relevant findings. Johan Margono and Lynn Lindsey, citing experience on the US Federal Aviation Administration's Advanced Automation System project, have shown that the relative cost of creating a reusable code component is about twice that of creating a nonreusable version, and the costs to integrate reused components into new products ranged from 10 to 20 percent of the cost of creating a nonreusable version.[3] John Favaro cites findings that the relative cost of producing a reusable component ranged from 120 to 480 percent of the cost of creating a nonreusable version, and integration costs ranged from 10 percent to 63 percent of the cost of creating a nonreusable version.[4]

Experience at HP in the STG graphics firmware domain has shown that the cost of creating reusable firmware is 111 percent of the cost of creating a nonreusable version, and integration costs were 19 percent of the cost of creating a nonreusable version.

Figure 4 shows the percent increase in engineering months by life-cycle phase (except maintenance) in creating a reusable software work product in the STG division. The data shows that the most significant increases were in the investigation and external design phases. This is because the producer of the work product required a greater amount of time to understand the multiple contexts in which the work product will be reused. Margono has also cited additional costs by life cycle phase in producing reusable code.[5] He found that analysis and top-level design

**TABLE 2
COST TO PRODUCE AND REUSE**

Domain	Air-traffic-control system	Menu- and forms-management system	Graphics firmware
Relative cost to create reusable code	200%	120 to 480%	111%
Relative cost to reuse	10 to 20%	10 to 63%	19%

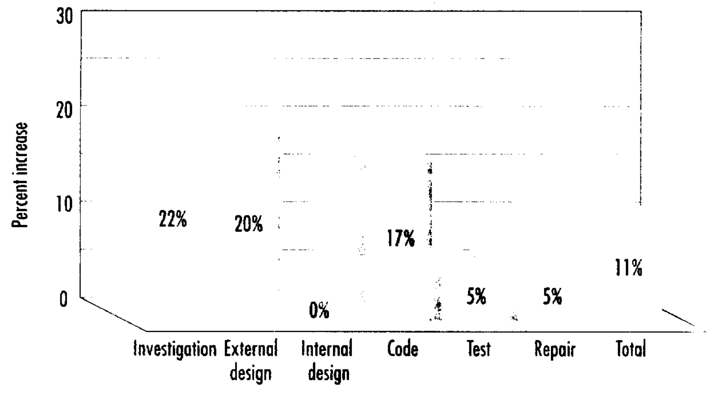


Figure 4. Additional effort in the San Diego Technical Graphics Division to create the reusable work product, by phase. Investigation is the initial analysis of user requirements, product risks, and benefits; external design is the detailed analysis of user requirements and definition of the product's external view; internal design is the translation of external design into detailed design of system and modules; code includes coding through unit testing; test is integration and system test through alpha and beta test; and repair is the repair of defects discovered during test phase. Maintenance is not included.

required 10 percent more than normally incurred in that phase (15 percent more for complex components), in the detailed design phase, 60 percent more, and in the code and unit test phases, 25 percent more.

REUSE ECONOMICS

An important aspect of software reuse is the economic return the organization receives for its efforts. Bruce Barnes, Terry Bollinger, Tom Durek, John Gaffney, and Shari Lawrence Pfleeger have done pioneering research in reuse economics. Gaffney and Durek present a relative cost model that describes development with reuse as a proportion of a baseline project.[6] Barnes, Bollinger, and Pfleeger[7-8] determine the cost/benefit by subtracting the producer investment costs of making work products reusable from the consumer development costs saved net of adaptation costs.

The technique of economic analysis used for the two HP reuse projects is the well-established net-present-value method.[9] Net present value takes the estimated value of reuse benefits and subtracts from it associated costs, taking into account the time value of money.[10] This model contributes to the field of reuse economics by recognizing the potential increased profit from shortened time-to-market and accounting for risk. Because the economic benefit derived from shortened time-to-market is difficult to assess, the overall economic benefit shown for each HP reuse program is conservative. The model is also meant to be applied over the entire life cycle, including maintenance.

An economic analysis may be performed for a reuse program or a given reusable work product. We begin with economic analyses at the program level.

Program savings. Figure 5a shows the economic analysis for the MP section's reuse program calculated over 10 years; Figure 5b shows the same type of analysis for the STG division's program over eight years (the final year uses estimated costs and benefits). Table 3 summarizes the experience of these two projects. To account for the time value of money, we discounted the cash flows at a 15 percent discount rate.

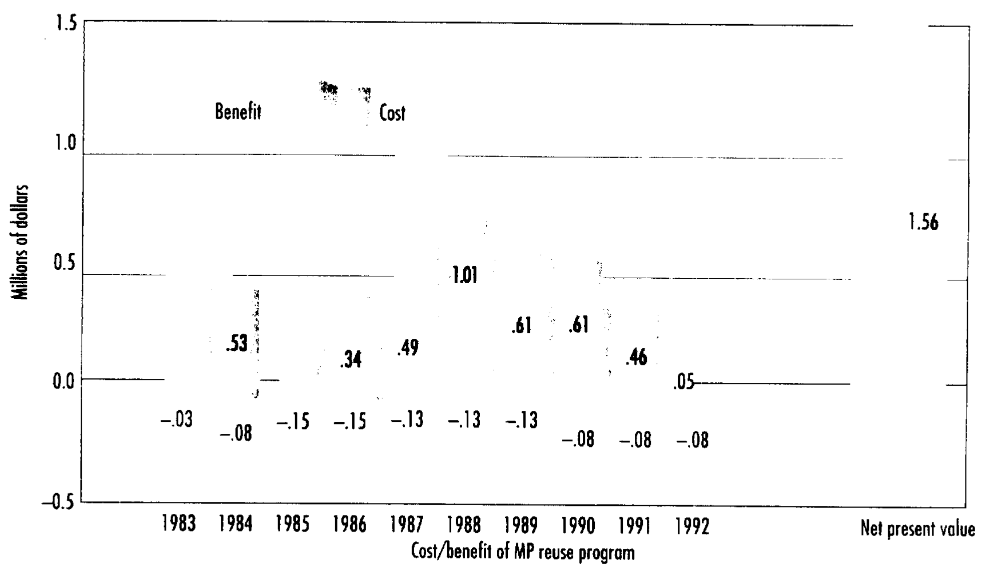
By creating reusable work products periodically as the opportunity arose, the MP section pursued an incremental investment strategy. Its reuse program required about 26 engineering months (about $0.3 million) as startup expenses, including work-product creation and engineer training, for six products. Because no reuse-specific tools were purchased, engineers' time constituted the majority of the expense. Ongoing expenses, such as the maintenance of the reusable work products, were about $0.7 million, for gross expenses of about $1.0 million. The gross savings during this period was $4.1 million, for a return on investment of 410 percent. A net-present-value analysis indicates a savings of $1.6 million. The break-even point occurs in the second year.

The STG division devoted the time of three engineers for three years to create its reuse work product, which was used by development projects in subsequent years. Startup resources — again, mostly engineers' time — were $1.3 million, and ongoing expenses were also about $1.3 million. The gross costs of $2.6 million are offset by gross savings of $5.6 million. So the return on investment is 216 percent, and the net present value is $0.9 million. The break-even point occurs in the sixth year.

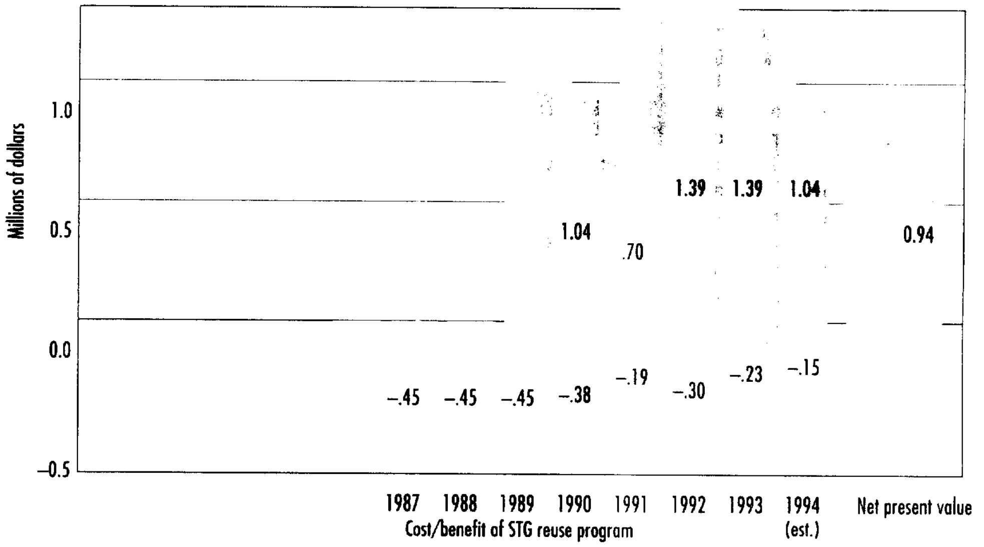
To compare the economic performance of the two reuse programs, we also determined the MP section's return over eight years (1983-1990). Its return on investment for this time period was 422 percent, with a net present value of 1.4 million. These figures can be directly compared to the STG division's return of 216 percent and net present value of $0.9 million.

TABLE 3
REUSE PROGRAM ECONOMIC PROFILES

Organization	Manufacturing Productivity	San Diego Technical Graphics
Time horizon	1983 - 1992 (10 years)	1987 - 1994 (8 years) 1994 data estimated
Start-up resources required	**26 engineering months (start-up costs for six products) $0.3 million**	**107 engineering months (about three engineers for three years) $0.3 million**
Ongoing resources required	54 engineering months (about one-half engineer for nine years) About $.3 million	99 engineering months (about one to three engineers for five years) About $0.7 million
Gross cost	**80 engineering months ($1.0 million)**	**206 engineering months ($2.6 million)**
Gross savings	328 engineering months ($4.1 million)	446 engineering months ($5.6 million)
Return on investment (savings/cost)	**410%**	**216%**
Net present value	125 engineering months ($1.6 million)	75 engineering months ($0.9 million)
Break-even year (recoup start-up costs)	**Second year**	**Sixth year**



(A)



(B)

Figure 5. Reuse cost-benefit analysis by program. (A) Manufacturing Productivity section over 10 years; (B) San Diego Technical Graphics Division over eight years.

Product savings. Figure 6a shows an analysis of 15 reusable work products — ranging in size from 58 to 2,257 NCSS — created in the MP section, indicating the savings a consumer receives from reuse. In the MP section, the savings ranged from 0.2 to 4.0 engineering days. Figure 6b shows an economic analysis from the perspective of the producer for the same 15 work products. This analysis attempts to answer the question, "Is it worthwhile for me as a producer to create this reusable work product?" The results range from a gain of 43.3 engineering days to a loss of 31.5 engineering days (the 31.5 engineering-day loss was the result of fewer than expected reuses and higher than expected maintenance costs). These figures do not include overhead costs, such as the manager's time, only the direct time spent by the producer to create the reusable work product.

Figure 7 shows the analysis from the perspective of the STG consumer at the work-product level. In the first year, the time required for the consumer to understand, adapt, and integrate the reusable work product is \$0.07 million. The savings to the consumer in the initial year from not having to create the functionality that the reusable work product provides is \$0.36 million. In the second year, the consumer avoids having to repair defects that he would have otherwise had to had he created the functionality from scratch. This cost avoidance is \$0.06 million. Using a 15 percent discount rate to take into account the time value of money, the net value received by a consumer with each reuse of this work product is \$0.35 million.

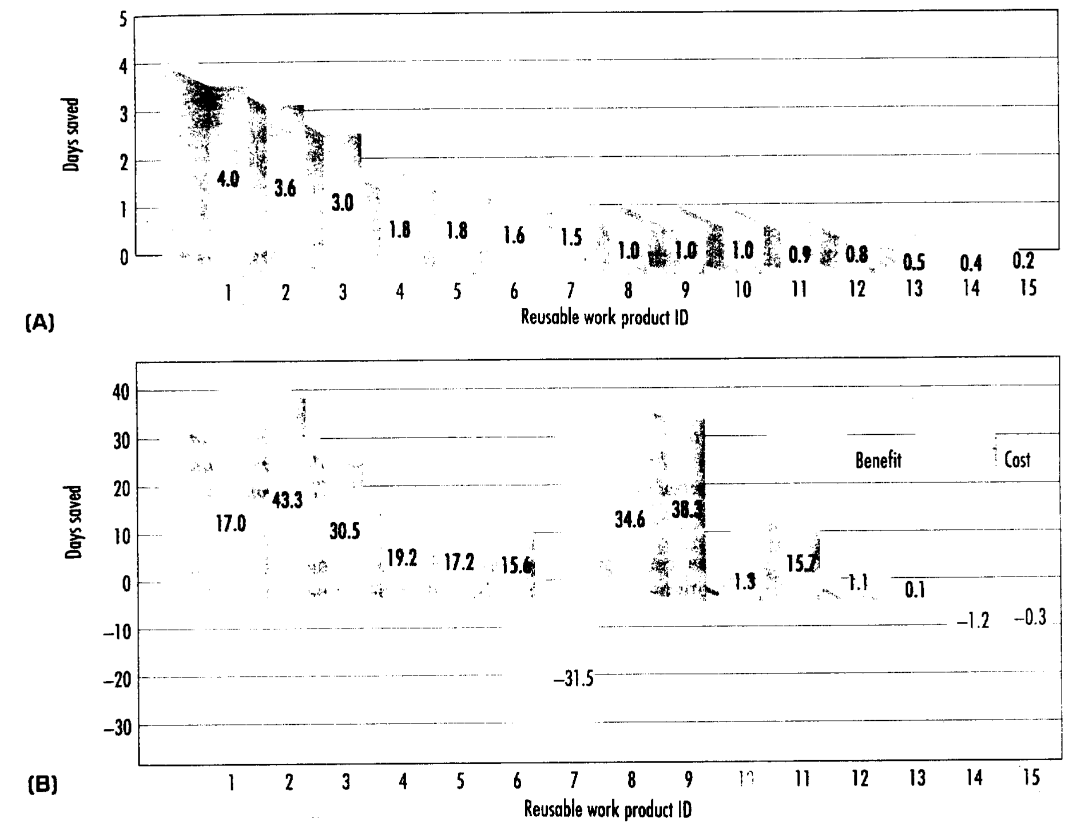


Figure 6. Net present value of 15 work products — as measured in days saved — from the Manufacturing Productivity section. (A) NPV received by work-product consumer; (B) NPV provided by work-product producer to the organization.

DECISION SUPPORT

You can also use this economic model to determine the economic viability for work products under consideration to be made reusable. Figure 8 displays the results of such an analysis for four potential work products in the MP section. These results were determined using the same cost-benefit model described earlier. Such an analysis helps personnel decide which work products are economically worthwhile to create and, for resource-constrained producers, the sequence in which they should be created.

In this example, all four work products are economically worthwhile to create because their net present values are positive. The areas of the circles are proportional to their net present values. The number of reuses to break even (recover creation costs) ranges from one to eight. An economic ranking of these work products suggests that producers create the work products in the following sequence: 1, 2, 4, and 3. In prioritizing the creation of reusable work products, producers should also take into consideration other factors such as the schedules of the consumer projects.

While the overall economics for the two reuse projects have been positive, economic analysis for one of the programs indicates that creating some of the work products has resulted in an overall economic gain and a few have resulted in a loss. Performing cost-benefit analyses for potential work products helps determine which work products should be created or reengineered to be reusable.

The data collected for these two programs have been used to initiate

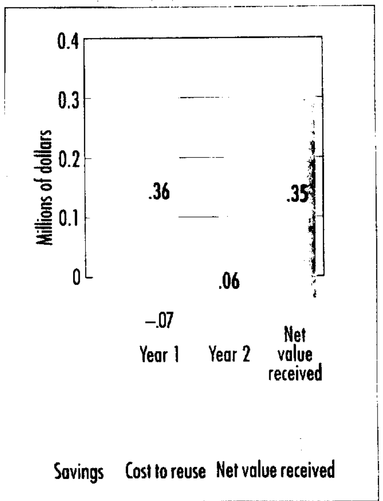


Figure 7. Net present value — as measured in millions of dollars — received by the consumers of the work product from the San Diego Technical Graphics Division.

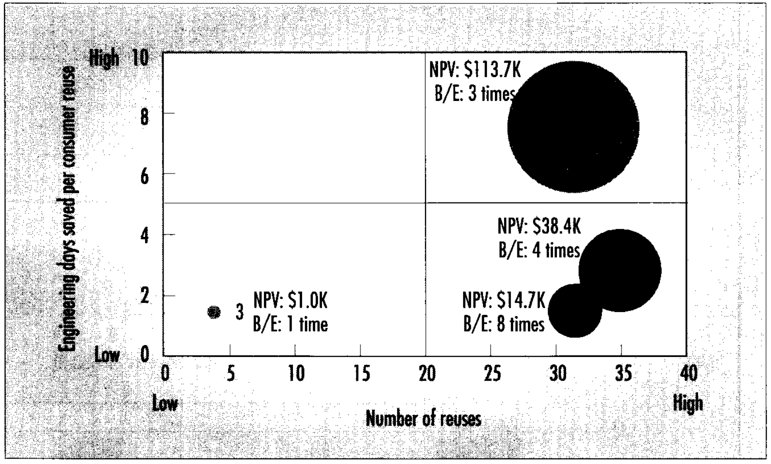


Figure 8. *A "reuse value map" of work products under consideration to be designed for reuse in the Manufacturing Productivity section. In this case, the total potential savings – the portfolio NPV– is $167.7 thousand.*

other reuse programs across HP. In addition, the results have been distributed as an example of start-up and ongoing costs and benefits of reuse for managers considering reuse in their divisions. ◆

ACKNOWLEDGMENTS

I thank Jack Cassidy, Sylvia Kwan, Joe Mueller, Alvina Nishimoto, Masahiro Yokokawa, and their teams for their help with collecting data and permission to publish this information. Additional acknowledgments to the HP Corporate Reuse team for their comments on previous versions of this article. Earlier versions of this work were presented at the 1992 Pacific Northwest Quality Conference and the 2nd International Workshop on Software Reuse.

REFERENCES

1. A. Nishimoto, "Evolution of a Reuse Program in a Maintenance Environment," *Proc. 2nd Irvine Software Symp.*, University of California, Irvine, Calif., 1992, pp. 89-108.
2. M.D. Lubars, "Affording Higher Reliability Through Software Reusability," *Software Eng. Notes*, Oct. 1986, p. 39.
3. J. Margono and L. Lindsey, "Software Reuse in the Air Traffic Control Advanced Automation System," *Proc. Software Reuse and Reengineering Conf.*, Nat'l Inst. for Software Quality and Productivity, Washington, D.C., 1991.
4. J. Favaro, "What Price Reusability," *Proc. First Symp. on Environments and Tools for Ada*, ACM, New York, 1990, pp. 115-124.
5. J. Margono, and T. Rhoads, "Software Reuse Economics: Cost-Benefit Analysis on a Large-Scale Ada Project," *Proc. Int'l Conf. Software Eng.*, CS Press, Los Alamitos, Calif., 1992, pp. 338-348.
6. J.E. Gaffney and T.A. Durek, "Software Reuse — Key to Enhanced Productivity: Some Quantitative Models," Tech. Report SPC-TR-88-015, Software Productivity Consortium, Herndon, Va., 1988.
7. B.H. Barnes and T.B. Bollinger, "Making Reuse Cost-Effective," *IEEE Software*, Jan. 1991, pp. 13-24.
8. T.B. Bollinger and S.L. Pfleeger, "The Economics of Software Reuse," Tech. Report CTC-TR-89-014, Contel Technology Center, Chantilly, Va., 1989.
9. W. Lim, "A Cost-Justification Model for Software Reuse," *Proc. 5th Annual Workshop for Institutionalizing Software Reuse*, University of Maine, Orono, 1992.
10. R. Brealey and S. Myers, *Principles of Corporate Finance*, McGraw-Hill, New York, 1981.

Wayne C. Lim specializes in the strategic planning, economic, organizational, and metrics issues of software reuse. The analysis he presents in this article is the result of his work while he was at Hewlett-Packard, where he managed reuse organizational and economic assessments. He is the author of *Managing Software Reuse* (Prentice-Hall, 1995, to appear).

Lim received a BA in mathematics from Pomona College and an MBA from Harvard University.

Address questions about this article to Lim at 1535 Fairway Green Cir., San Jose, CA 95131; lim@source.asset.com.

Free report from Peter Coad reveals amazing industry breakthrough!

"Object modeling and C++ programming, side-by-side, always up-to-date."

Big CASE tool vendors caught with their pants down!

What if you could have your OOA/OOD model and all of your C++ code continuously up-to-date, all the time, throughout your development effort?

Consider the possibilities...

In one window, you see an object model, with automatic, semi-automatic, and manual layout modes, plus complete view management. Side-by-side, in the other window, you see fully-parsed C++ code. You edit one window or the other. Press a key. Both windows agree with each other. **Together.**

Suppose that you are working on a project with some existing code. (That's no surprise, who'd consider developing in C++ without some off-the-shelf classes?) You read the code in. Hit a button. And seconds later, you see an object model, automatically laid out and ready for you to study side-by-side with the C++ code itself. **Together.**

Or suppose you are building software with other people (that's no surprise either). You collaborate with others and develop software with a lot less hassle, because the fully integrated configuration management feature helps you keep it all...**Together.**

The name of this product? It's earned the name...

Together/C++
continuously up-to-date
object modeling and C++ programming

Call now for your free report!

Key features:
- Continuously up-to-date object modeling & C++ programming
- Automatic, semi-automatic, and manual layout of object models
- Object modeling view management, including view control by C++ construct, regular expression, proximity, layer, or directory
- Fully flexible documentation generation, version control, and SQL generation

"State-of-the-art application development."
-- *Computerworld*/Germany

"You've really hit the nail on the head when it comes to reverse engineering existing C++ code. No other tool comes close to the power and capability of Together/C++."
-- Russell Rudduck, Perot Systems

Money-back gurantee. Purchase Together/C++ and try it out risk-free for 30 days. If for any reason you aren't satisfied, return it for a full refund. (No hassles, no hard feelings either.)We're that confident about Together/C++. You see, Together/C++ has already helped software developers deliver better systems, with success stories in telecommunications, insurance and natural resource managment.

How to order. Order Together/C++ by purchase order, check, or credit card, or for more information, please contact:

Object International, Inc.
Education - Tools - Consulting
8140 N. MoPac 4-200
Austin TX 78759 USA
1-800-OOA-2-OOP
1-512-795-0202 - fax 795-0332
e-mail: object@acm.org

Outside of North America, contact:
Object Int'l Ltd.
Eduard-Pfeiffer-Str.73
D-70192 Stuttgart, Germany
++49-711-225-740 - fax 299-1032
100034.1370@compuserve.com
©1994 Object Int'l, Inc.
All rights reserved.
"Together" is a trademark of Object Int'l, Inc.

IEEE994