# Applying an Improved Economic Model to Software Buy-versus-Build Decisions

The decision to buy or build software is a business decision that should be made using a sound economic model. A comprehensive economic model has been developed and applied to actual and estimated data to compare the costs of using a third-party software package to the costs of internal development.

**by Wesley H. Higaki**

Hewlett-Packard Laboratories has been recommending the use of third-party software packages to product divisions whenever it makes sense economically. We feel that those divisions whose primary products are not databases, networks, or operating systems should purchase these technologies. There are many readily available, shrink-wrapped, off-the-shelf products. Hewlett-Packard Company offers many application-specific software products such as electronic instruments and measurement systems. The product divisions that offer these applications can be more productive if they focus their efforts on the applications rather than devoting engineering resources toward developing technologies that are already available from third-party vendors.

There are, of course, valid reasons for not using third-party software in an application. Such software should not be used if the third-party product does not meet the application's specific functional requirements, if the third-party vendor is not reliable, financially stable, or responsive enough to customer needs, or if the third-party product costs too much.

While there are valid reasons not to use third-party software, there seems to be a bias in the engineering community towards building rather than buying software. The decision to buy or build software is a business decision that should be made using a sound economic model. Without this model and a complete analysis, inferior business decisions may be made.

This article applies a comprehensive economic model to actual and estimated data to compare the costs of using a third-party software package to the costs of internal development.

### Past Economic Models

Today, when product teams do an economic evaluation to build or buy software, they sometimes make assumptions that can lead to less than optimal business decisions. Some of these assumptions are:

- *Initial software development costs are sunk.* The common view is that development engineers are already paid for, so it is always cheaper to write code internally than to buy it from the outside. Essentially, internal software development costs are considered zero. This is an indication that the return-on-investment model used today is flawed.
- *Ongoing maintenance costs are invisible.* Engineering effort will be expended over time to repair defects, add product enhancements and maintain compatibility with system upgrades. These costs, like initial development costs, are not visible and not accounted for in calculating returns.
- *Lost opportunities are not accounted for.* While development engineers are developing what could be bought, they are missing opportunities to add application value to the product. Development engineers may also help get their product to market earlier by not writing code from scratch.
- *Licensing costs are added to manufacturing costs and passed on directly to the customer.* This is a flaw in the software pricing model. If code is developed internally, the product is priced on market value. If code is bought and a license fee is attached, the licensing fee is generally considered a manufacturing cost.

### An Improved Economic Model

To address the issues in the current economic models used, I propose the use of an improved model. There are four parts to this improved economic model for evaluating build-versus-buy decisions. These parts are:

- The Malan/Wentzel model
- An extensive list of costs and benefits of buying software
- Net present value calculations
- Estimation techniques for costs and benefits.

This improved economic model enables us to examine the hidden costs and benefits of buying software. We can then analyze the economic effects of buying more completely.

### Malan/Wentzel Model

HP Laboratories has been researching models that evaluate the economics of software reuse. Using packaged software is a form of software reuse. The Malan/Wentzel model[1] produces a comprehensive picture of the true costs of software development and the savings that result from reuse. This

model is based on development phase costs, maintenance phase costs, and other reuse-related benefits and costs. Its developers point out that the savings in the development phase represent a conservative estimate of the benefits of reuse. The complete model takes into account the effects of increased profits resulting from early time to market and the exploitation of new opportunities.

### List of Benefits and Costs

The following is a modified list from Poulin, Caruso, and Hancock[2] showing the benefits and costs of reusing packaged software. This list includes the benefits and costs of buying software through the development and maintenance phases. By using this list in conjunction with the Malan/Wentzel model, we can do a more complete analysis of the costs involved in making a buy-versus-build decision. In this discussion, a component is a piece of a software product whose functionality can potentially be filled by packaged software. This component may be purchased or it may be built.

Listed in Table I are the cost savings that may result during the initial development and maintenance phases if a component is bought rather than built.

**Table I**
**Benefits of Buying**

| |
|---|
| **Initial Development Savings** |
| Reduced cost to design the component |
| Reduced cost to implement the component |
| Reduced cost to test the component |
| Reduced cost to document the component |
| **Ongoing Maintenance Savings** |
| Reduced cost to fix defects in the component |
| Reduced cost to enhance the component |

Buying software saves development time and thus the product can be delivered earlier than if all of the software is written from scratch. Earlier time to market can result in increased profits from two effects: added profit from delivering the product earlier to the marketplace, and added profit from increased market share over the life of the product.

It also costs something to buy software. The process involved in the build-versus-buy decision for software as described by Malan and Wentzel has four steps:

- Define the requirements of the component.
- Search for and acquire the component, determine what the component does, and verify that it meets the requirements.
- Integrate the component into the rest of the product.
- Customize the product to meet the specific application requirements.

There are costs associated with each of these steps relative to buying software. Table II shows a more detailed list of these costs.

### Net Present Value Calculations

Since benefits and costs occur at different times in the product's life cycle, the time value of money must be taken into account. The net present value (NPV) equation[2] calculates the true net benefit of software reuse.

$$NPV = \sum_{i=0}^{n} (B_i - C_i)/(1 + k)^i,$$

where n is the number of years in the product's life cycle, k is the annual discount rate, $B_i$ is the benefit realized in year i, and $C_i$ is the cost incurred in year i.

**Table II**
**Costs of Buying**

| |
|---|
| **Acquisition Costs** |
| Licenses or royalties for the third-party package |
| **Customization Costs** |
| Cost of customizing the third-party package |
| Cost of maintaining the customized component |
| **Assessment and Integration Costs** |
| Cost of performing the cost-benefit analysis |
| Cost of locating and assessing the third-party package |
| Cost of integrating the third-party package |
| Cost of training on the third-party package |

### Benefits of Buying

**Initial Development Savings.** The most immediate benefit of buying software over building it is that the initial engineering effort and costs are saved. This saving can be measured directly, based on the estimated engineering cost to build the component. This cost can be estimated using schedule estimates to design, develop, and debug the component multiplied by the run rate for the engineers involved.

**Ongoing Maintenance Savings.** Harris[3] estimates that software maintenance costs for internally developed software are about 55% of the total cost of the product life cycle. This estimate includes costs associated with fixing defects as well as implementing enhancements. We can estimate the reduced maintenance costs using the actual initial development costs of the internally developed software by the following formula:

$$C_M = (0.55/0.45)C_D,$$

where $C_M$ is the cost of maintenance over the life of the product and $C_D$ is the initial development cost of the product. Thus, using the estimate from the initial development, we can calculate the expected ongoing maintenance costs.

**Increased Profits.** Patterson[4] states that for every month a product release is delayed, one month of sales is lost. Conversely, delivering a product early results in increased sales. Patterson asserts that once a product with a given set of capabilities is released, the date the product will become obsolete is set. Thus, if two products are released at different times with the same capabilities, they will both be obsolete on the same date because of marketplace and competitive pressures. If these two products are released on different dates, then the product released earlier will have the greater sales. Fig. 1 illustrates the effect of an early

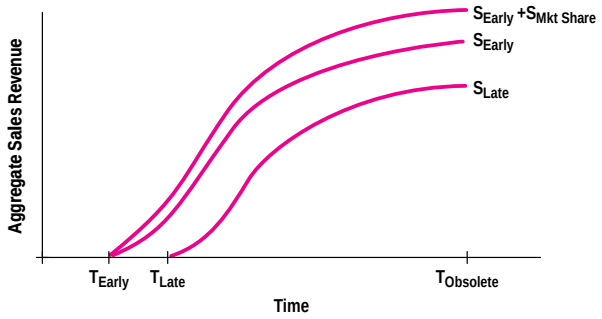


**Fig. 1.** Effect on sales of earlier time to market. Curve $S_{Early}$ represents the sales of the product introduced at time $T_{Early}$. The curve $S_{Late}$ is the sales curve for the product introduced at time $T_{Late}$. The curve $S_{Early} + S_{Mkt\ Share}$ illustrates the total effect on sales of more time in the marketplace and increased market share.

product introduction on sales revenues. Curve $S_{Early}$ represents the sales of the product introduced at time $T_{Early}$. The curve $S_{Late}$ is the sales curve for the product introduced at time $T_{Late}$.

Software reuse can improve a product's time to market. If the product reaches the market two months early with the same functionality, then there will be an additional two months' revenue. To estimate this effect, we will use the projected annual sales and estimate the improvement in time to market.

$$S_I = S_V T_M,$$

where $S_I$ is the sales increase resulting from early time to market, $S_V$ is the original projected volume per month, and $T_M$ is the improvement in time to market in months.

Moreover, Smith and Reinertsen[5] suggest that market share increases over the life of the product as a result of earlier time to market. This is expressed as an increase in market share percentage throughout the life of the product. This estimate is based on competition, market demand, and time to market. This effect can be estimated by using market research data to determine how sales volumes would be affected by providing features earlier than the competition. This can be expressed as more units sold per year or per month. The curve $S_{Early} + S_{Mkt\ Share}$ in Fig. 1 illustrates the total effect on sales of more time in the marketplace and increased market share.

### Costs of Buying

**Acquisition Costs.** Sometimes there are up-front acquisition costs for third-party software. More typically, there are licensing costs or royalties associated with using a software package. These costs can be easily estimated by using sales volume estimates for the end product and determining how many copies of the third-party package will be required.

**Customization Costs.** This model assumes that no modification of the third-party software is required to meet the product requirements. However, customization may be required. An example of such customization is using Microsoft® Excel to build a spreadsheet that calculates car payments. The customization is the act of developing the spreadsheet to do the appropriate calculations. The Excel product itself is not modified. The customization costs are derived by estimating the engineering effort for customizing the third-party package.

**Assessment and Integration Costs.** To determine if in fact a third-party package will meet the product requirements, some engineering evaluations must take place. Estimates of how much effort and cost are involved in the evaluation, assessment, and integration of the third-party package into the application can be made.

### Case Study

In a project that HP Laboratories recently completed with an HP product division, the product team made extensive use of third-party tools and components, but chose to build a report writer subsystem rather than buy a report writer package. The product required that reports be generated from data stored in a dBASE IV® database on a PC running Microsoft Windows 3.1. Several off-the-shelf packages were available ranging in price from $200 to $2,500 per copy (all amounts in this paper are in U.S. dollars). One package seemed to meet the requirements and cost $400 per copy, but the product team decided to develop the report writer internally. This decision was made based on the assumption that $400 per copy was more than what it would cost to build the report writer internally.

The report writer is only one component in the product. If the recommended third-party package had been used in place of the internally developed report writer, the project would have been completed about two months earlier. This third-party package is a tool that enables the developer to create custom reports from data stored in dBASE IV. It would take an engineer about one week to learn this package, another week to develop the report formats, and one more week to integrate it into the rest of the product. An ongoing enhancement effort of about one week per year to add new report formats is expected to continue throughout the product's life.

Table III summarizes the costs and benefits determined using the economic model to compare the use of this third-party package with an internally developed report writer. According to the model, the net savings of buying the software in this case would have been $213,754 over the life of the product.

**Initial Development Savings.** The actual time it took to design, implement, integrate and test the report writer for this project was one engineer-year. A fully loaded engineer costs the company about $100,000 per year. So, the savings in initial development would be $100,000. These savings would have been realized in the year before the product release.

**Ongoing Maintenance Savings.** Using the maintenance cost estimate formula, the estimated maintenance cost for the lifetime of the product is:

$$C_M = (0.55/0.45)(\$100,000) = \$122,222.$$

Assuming this cost is evenly distributed over the four-year life of the product, the cost is about $30,500 per year.

Introducing the time value of money at a 6% discount rate results in a $105,686 benefit.

**Table III**
**Cost-Benefit Analysis**

| Benefits | Savings or (Costs) |
|---|---|
| **Initial Development Savings** | $100,000 |
| Reduced cost to design | |
| Reduced cost to implement | |
| Reduced cost to test | |
| Reduced cost to document | |
| **Ongoing Maintenance Savings** | $105,878 |
| Reduced defect fixing costs | |
| Reduced enhancement costs | |
| **Increased Profits** | $72,387 |
| Added profit from delivering product sooner to the market | |
| **Costs** | |
| **Acquisition Costs** | ($41,581) |
| Licenses or royalties for reusing parts | |
| **Customization Costs** | |
| Cost of customizing the third-party package | ($2,000) |
| Cost of maintaining the customized component | ($6,930) |
| **Assessment and Integration Costs** | |
| Cost of performing the cost-benefit analysis | ($5,000) |
| Cost of locating and assessing the third-party package | ($5,000) |
| Cost of integrating the third-party package | ($2,000) |
| Cost of training on the third-party package | ($2,000) |
| **Net Savings** | $213,754 |

The maintenance effort includes not only defect fixes,[6] but also updates for new fonts, printers, and report formats. The internally built report writer was only a set of report templates and not a template generation tool like the recommended third-party package. This means that whenever there is a request for a new report template, more software needs to be written. With the third-party package, a new template can be created quickly. Since much of the division's sales are for custom systems, the Harris estimate[3] seems to be consistent with this example.

There are also quality and competitive issues involved with developing an internal report writer. Since the third-party company is in the business of developing and selling report generators, they (and their competitors) will invariably do a more complete job of developing report generators than this HP division will. The HP division will be forced to compete with them as customers demand greater functionality to match that offered in packaged report writers and this will increase the enhancement effort.

**Increased Profits.** Based on projected sales for this product and the anticipated earlier product introduction, the division would realize $500,000 additional revenue. It is also estimated that there would be more units sold per year through out the entire product life because of the earlier initial product introduction and the availability of engineering resources to develop value-added features. Using an appropriate profit margin on the increased sales volume and applying the NPV calculation yields the $72,387 benefit shown in Table III.

**Acquisition Costs.** There are no fees associated with the initial acquisition of the third-party product, only licensing fees of $400 per copy. Applying the NPV formula to the estimated unit sales for the life of the product results in a $41,581 cost to pay for the licenses.

**Customization Costs.** While no changes need be made to the third-party package, report templates would have to be generated for the HP product. The effort required to generate the initial report templates and to integrate the report generator with the rest of the application is estimated to cost about $2,000. The engineer would have to develop the report formats using the third-party package. It is also estimated to cost about $2,000 per year to add any new report formats requested by customers. The NPV formula is applied to these costs for the life of the product to yield $6,930.

**Assessment and Integration Costs.** It is estimated that an engineer would cost $10,000 to do the domain analysis, locate and assess the packaged software, and do a cost-benefit analysis. It is also estimated to take another $2,000 to integrate the package into the rest of the HP product.

The cost of learning the third-party product is estimated to be about $2,000.

**Cost-Benefit Analysis.** First, we compare just the costs of development with the licensing costs. The total cost of developing and supporting the report writer is about $206,000 ($100,000 initial development plus $106,000 for ongoing maintenance). Divide this by the number of units expected to be sold during the product's life, and the cost per copy of this report writer is estimated to be over $1,000. These costs far exceed the $400 per copy licensing fee for the third-party package. This alone indicates a decision to buy rather than build.

To do the complete analysis, increased profit factors and package assessment costs are introduced. The net benefit of using the third-party package rather than developing a report writer is $213,754.

The greatest benefit comes from delivering the product early. Arguments from Stalk and Hout,[7] Patterson,[4] and Smith and Reinertsen[5] list the advantages of delivering products early, and this model demonstrates the impact time to market has on a product's revenues. The time-to-market benefits dwarf the $400 per copy licensing costs of the packaged software.

## Conclusions

The product team thought they would have to add the $400 cost of the third-party product as a separate line item on the customers' purchase orders. They were concerned that this would confuse the customer, causing discussions that could hinder the sale and reduce overall sales. If the third-party package were bundled with the product so that the customer could not see what was included in the product, there could have been no objections. For accounting purposes, this $400 cost could have been recorded as an R&D expense and not a manufacturing expense since manufacturing expenses automatically have multipliers attached. The total

price would remain the same, but profit would be greater because costs would be lower.

### Risks

While the focus of this article has been on the cost-benefit analysis of buy-versus-build decisions, this analysis is useful only after the preliminary evaluations have taken place and the package has been proven to meet the requirements of the product. These requirements must be carefully evaluated first.

Product quality and reliability are especially important to HP, a company for which quality is a major differentiator in the marketplace. The packaged software must meet the functional and quality requirements before any further consideration. Using a packaged product that comes from a proven vendor and has a history in the marketplace of providing complete functionality and solid support mitigates the risk of choosing an inadequate package.

Another major risk is the longevity of the vendor. No one wants to rely on a package that is no longer supported by the vendor. Again, using packages from proven vendors will help reduce this risk. Also, choosing components that are supported by a number of qualified packages from competing vendors allows the replacement of one package with another in case the initial vendor cannot maintain the package any longer.

There are several risks involved in developing software internally. The more obvious risk is the effort required for defect fixing. There is also the risk that internally developed software will have to compete with other packaged software. This is a battle that cannot be won, since report writer vendors, for example, expend much more effort developing report writers than does a product team trying to deliver a measurement system.


### Acknowledgments

Many thanks to Ruth Malan of HP Laboratories, who has encouraged me, helped direct me to the proper sources, and provided inputs on the approach to take in presenting my ideas.


### References


1. R. Malan, K. Wentzel, *Economics of Software Reuse Revisited*, HP Laboratories Technical Report HPL-93-31, April 1993.
2. J.S. Poulin, J.H. Caruso, and D.R. Hancock, "The Business Case for Software Reuse," *IBM Systems Journal*, Vol. 32, no. 4, 1993.
3. K. Harris, "Using an Economic Model to Tune Reuse Strategies," *Proceedings of the Fifth Annual Workshop on Software Reuse*, 1992.
4. M. Patterson, *Accelerating Innovation*, Van Nostrand Reinhold, 1993.
5. P.G. Smith, and D.G. Reinertsen, *Developing Products in Half the Time*, Van Nostrand Reinhold, 1991.
6. W.T. Ward, "Calculating the Real Cost of Software Defects," *Hewlett-Packard Journal*, Vol. 42, no. 4, October 1991, pp. 55-58.
7. G. Stalk, Jr. and T.M. Hout, *Competing Against Time*, The Free Press, 1990.